IESD-COMP-1171

Summary Report
AMTRAN Development Program

AMTRAN SYSTEM DESIGN AND SOFTWARE DESCRIPTION

by M. E. Dyer

May 1970

TELEDYNE
BROWN ENGINEERING

Research Park • Huntsville, Alabama 35807

SUMMARY REPORT
IESD-COMP-1171

AMTRAN DEVELOPMENT PROGRAM

AMTRAN SYSTEM DESIGN AND SOFTWARE DESCRIPTION

By

M. E. Dyer

May 1970

Prepared For

COMPUTATION LABORATORY
GEORGE C. MARSHALL SPACE FLIGHT CENTER
HUNTSVILLE, ALABAMA

Contract No. NAS8-20415

Prepared By

SCIENCE AND ENGINEERING
TELEDYNE BROWN ENGINEERING
HUNTSVILLE, ALABAMA

ABSTRACT

This report contains a detailed description of the software developed by Teledyne Brown Engineering to implement the AMTRAN (Automatic Mathematical Translation) language on the IBM 1130 computer with 8K of core. Included are a description of the language and full documentation of the software system.

Approved:

R.R. Parker

R. R. Parker, Ph.D.
Manager
Data Processing Laboratories

TABLE OF CONTENTS

TABLE OF CONTENTS - Continued

TABLE OF CONTENTS - Continued

TABLE OF CONTENTS - Continued

Page

TABLE OF CONTENTS - Concluded

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS - Concluded

LIST OF TABLES

1. SYSTEM DESCRIPTION

Automatic Mathematical Translation (AMTRAN) is a conversational mode computing language oriented toward the solution of mathematical problems. The personnel of Brown Engineering Company, Inc., have implemented a version of AMTRAN on a small general purpose computer providing conversational interaction with the user through the console keyboard/printer.

The AMTRAN system features a well defined, easily learned syntax and provides automatic array arithmetic and dimensioning of variables. In the conversational mode, as each AMTRAN statement is entered into the system, it is interpretively executed. The system performs checks for syntax and execution errors. The features of automatic array arithmetic and automatic variable dimensioning allow the user to construct generalized algorithms which do not require alterations when applied to new data sets. Algorithms and functions may be constructed by the user and stored by name in the system. User programs may be constructed in the conversational mode or entered into the system for later use without being executed as they are entered. At any time, user programs may be called for execution or may be altered or deleted by the user. The AMTRAN language provides a clear, concise method for declaring parameters and passing parameters to programs and functions.

The AMTRAN software system provides dynamic storage allocation for data and user defined programs and functions, is modularized into logical core loads, contains a disk core overlay system, and is designed to facilitate extensions to the system.

1.1 AMTRAN CAPABILITIES

The AMTRAN system recognizes 50 characters and 56 reserved words. These provide the user with the following five types of statements:

- Assignment
- Program logic control
- Program call
- Input/output
- System control and utility.

The user is not restricted to entering a single statement at a time but may combine any of the first four classes of statements to form a larger programming unit. When combined, these statements are referred to as substatements and are separated by commas.

Computation is specified by arithmetic expressions designed to resemble algebraic expressions. The arithmetic operations provided are

- Exponentiation
- Negation
- Multiplication
- Division
- Addition
- Subtraction
- Concatenation
- Less than
- Greater than
- Equal
- Not equal
- Less than or equal
- Greater than or equal.

The following standard arithmetic functions may be used:

- Trigonometric sine
- Trigonometric cosine
- Natural logarithm
- Square root

- Arctangent
- Hyperbolic tangent
- Absolute value
- Exponential.

Computation is performed in floating point. Arithmetic operands may be scalars or arrays, provided the dimensions are consistent. Additional arithmetic functions have been included in the system to facilitate array arithmetic. The array subscripting capabilities allow reference to a single element or to contiguous elements.

The dimension of a variable may be dynamic in a program. The automatic dimensioning of variables is provided by the assignment statement. A variable appearing to the left of the equal sign in the assignment statement will assume the dimension of the result of the computation designated by the expression to the right of the equal sign. Only the exact amount of storage currently required for a variable is allocated.

The control of the logical flow of a user algorithm is provided by an unconditional branch, an iterative statement, and an IF statement. The IF statement consists of a conditional, a then, and an else clause. The then and else clauses may consist of several substatements including additional IF tests.

The program call statements provide the communication between user programs.

The input/output statements allow the input and output of data and user programs. Data input may be in free form; however, program input and output and data output are in a standard format.

The system control statements allow the user to name, store, and delete programs, to save or delete data, and to change the mode of statement entry to the system.

1.2 GENERAL DESIGN CONCEPTS

A principal objective in the design of the AMTRAN software was to optimally divide internal storage between the system software and the user storage areas, in order to maintain an effective balance between execution speed and internal storage available to the user. To meet this objective, two major design concepts were implemented.

- The input statement in the AMTRAN language is translated to a concise form which is then interpreted and executed.
- The system software is modularized into logical core loads so that only the module or modules necessary for the completion of a specific task or set of tasks need be in core at any particular time during execution of the system.

The internal form in which an AMTRAN statement is executed is patterned after a machine language with distinct operator codes and none or several operands for each operator. The internal structure of a user program, whether stored or under construction, is in a standard format and is completely relocatable. When stored programs (stored on disk) are executed, it is this form of the program which is brought into core for execution, not the source form of the program which remains on disk.

The various modules of the software system may be grouped into the primary functional units shown in Figure 1-1 and described below.

- System Control - This portion of the system controls the overall flow of execution between the various modules of the system and regulates the disk core overlay system.
- Incremental Translation - This portion of the system scans and parses the input statement, performs syntax checks, and generates the executable interpreter instructions.
- Execution - This portion of the system performs interpretive execution of the generated interpreter instructions and the core storage allocation for data and for user programs when executed.

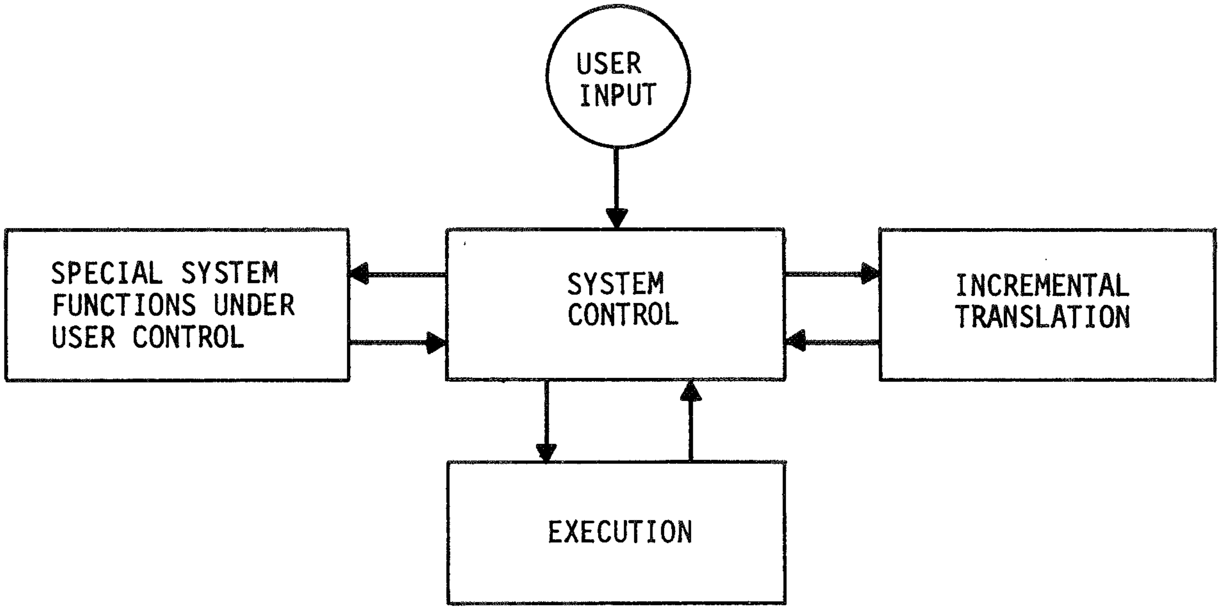


FIGURE 1-1. GENERAL SYSTEM ORGANIZATION

- Special System Functions under User Control - This portion of the system controls the storage, deletion, listing, and editing of user programs, the listing of user program names, the change of statement entry mode of the system, and the deletion or retention of user defined variables.

A major feature of the software system is the storage allocation for data. This allocation is a continuous process throughout execution of the interpreter instructions. Storage for user variables, temporary results, and the system accumulator is provided in one internal storage data area. Storage for any data type is allocated only as needed and in the exact amounts required during execution, is redimensioned as necessary, and, in the case of temporary storage, is made available for further use as soon as possible.

1.3 ENVIRONMENT AND LIMITATIONS

The AMTRAN system described is implemented on an IBM 1130 computer with 8K (16 bit words) of core storage. This computer contains a console typewriter/printer and a single disk drive and uses a card reader/punch and printer. The software system was written mainly in IBM 1130 FORTRAN IV with some 1130 Assembler language programs and is operational under Version II of the IBM 1130 Disk Monitor System.

Within the system, a user program must be written within the following limitations:

- Maximum of 45 statements (not to exceed 153 interpreter instructions)
- Maximum of 29 variables
- Maximum of 54 distinct constants, excluding the integers zero through ten
- Maximum of 10 distinct user programs called.

The system will allow the storage of a maximum of 95 user programs. User programs may be called 10 levels deep. Data storage for user defined variables, temporary storage, and the system accumulator contains 604 floating point words.

A version of AMTRAN is being developed on an IBM 1130 with 16k of core which provides the graphic display of data and alphanumeric information, an extended operator set, and relaxation of the restrictions imposed in the 8k implementation.

2. SYSTEM SPECIFICATIONS

The system specifications describe the final software system and include descriptions of the AMTRAN language, the design concepts, the functional organization of the software, and the system data base. Because the software was developed as a research project, the AMTRAN language was, in part, initially specified and was later extended as the software system was developed. A design concepts section is included in the specifications to present those concepts which underly the system software design, but which are not apparent in the functional organization of the system. The functional description of the system provides an overview of the functions performed by the system and of their interrelationships. The system data base is the collection of all data used by more than one program in the system.

2.1 INPUT/OUTPUT DESCRIPTION - AMTRAN LANGUAGE

This description is intended to present the basic concepts of the AMTRAN language and the characteristics of the AMTRAN syntax to allow understanding of the system software documentation. A detailed explanation of the AMTRAN language is provided by the users' manual ("AMTRAN Users' Manual", Brown Engineering Interim Report IESD-COMP-1103, March 1970).

The AMTRAN language allows the use of the character set {0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z * / + - ( ) . , & ; # $ blank} and a set of 56 reserved words. The alphanumeric characters are used to form numeric constants and labels. Numeric constants are entered as a string of digits with a minus sign to indicate a negative number. A decimal point is optional for integer quantities. An AMTRAN label consists of an alphabetic character followed by from zero to five characters which may be either alphabetic or numeric. No special characters or blanks are allowed in a label.

Labels are used as user program names, reserved system labels, and variable names which symbolically reference scalar or array quantities.

Labels, numeric constants, and special characters are combined to form AMTRAN program statements. An AMTRAN program statement consists of a decimal statement number which is output by the system and a string of input characters terminated by a period and an EOF. Each program statement may consist of one or more AMTRAN statements. The AMTRAN statements may be grouped into the following types:

- Assignment
- Program logic control
- Program call
- Input/output
- System control and utility.

With only a few exceptions, the user may combine statements of the first four types to form a program statement. When combined, these statements are referred to as substatements and are separated by commas.

2.1.1 Arithmetic Expressions

Computation in AMTRAN is specified by arithmetic expressions designed to resemble algebraic expressions. Arithmetic expressions are used in the assignment statement, the IF statement, and in several of the output statements. A wide range of computational capability is provided by the following types of AMTRAN operators:

- Arithmetic operations
- Mathematical functions
- Special operations for array construction and manipulation
- User defined arithmetic functions
- Subscripting
- Relational operations.

2.1.1.1 Arithmetic Operations - The following arithmetic operations are provided in AMTRAN:

- * or implied - multiplication
- / - division
- + - addition
- - - subtraction
- ** or POW - exponentiation.

The arithmetic symbol or label separates the two arguments. The operations can be performed between constants, variables, or expressions in the following combinations:

- Two scalars
- A scalar and an array of n elements
- An array of n elements and a scalar
- Two arrays, each with n elements.

An operation performed on the first combination will result in a scalar; an operation performed on each of the remaining combinations will result in an array of n elements.

The result of an operation between arrays is obtained by performing the operation between corresponding elements of the arrays. When one argument is a scalar, the operation is performed between the scalar and each element of the array. An error condition occurs when an arithmetic operation is attempted between arrays of different lengths.

2.1.1.2 Mathematical Functions - The following mathematical functions may be used in AMTRAN:

- SIN - trigonometric sine
- COS - trigonometric cosine
- LN - natural logarithm
- EXP - argument power of e
- SQRT - square root
- ATAN - arctangent
- ABS - absolute value
- TANH - hyperbolic tangent
- SQ - quantity squared.

Each of these functions requires one argument which may be either a constant, a variable, or an expression and which is entered after the label, except for SQ which requires the argument to precede the label.

The argument may be a scalar or an array; the function value will be a scalar or an array the same size as the argument. The result of a function performed on an array is obtained by applying the function to each element of the array.

2.1.1.3 Special Operations for Array Construction and Manipulation - The array arithmetic provided by the arithmetic operations and mathematical functions is augmented by the special operations described in Table 2-1.

2.1.1.4 User Defined Arithmetic Functions - A user may designate a sequence of statements to be a program using the NAME statement (see Table 2-3). To designate a functional program, the program name used in the NAME statement is the same as a variable defined in the program. A functional program is called for execution by entering the name followed by the parameters required. The value of the function is the current value of the corresponding variable when the return is made from the program.

2.1.1.5 Subscripting - Arrays are one-dimensional and are subscripted using the operator SUB to reference a single element or contiguous elements. The operator may be used in the following forms where x denotes an array variable or expression and a and b denote scalar constants, variables, or expressions:

- x SUB a - refers to the (a + 1)th element of x. (In AMTRAN, subscripting begins at zero.)

- x SUB (a THRU b) - refers to the (a + 1)th through the (b + 1)th elements of x.

- x SUB LAST or x SUB INTERVALS - refers to the last element of x.

2.1.1.6 Relational Operations - The relational operations

- LT - less than
- GT - greater than

TABLE 2-1. ARRAY CONSTRUCTION AND MANIPULATION OPERATIONS

Operation and Argument(s)[1]	Value(s) Returned
ARRAY a, b, c, or RANGE a, b, c	An array with the first element equal to a, the last element equal to b, and the number of equal sized increments equal to c
MIN x	A scalar which is the minimum valued element of x
MAX x	A scalar which is the maximum valued element of x
SUM x	An array the same size as x which contains the running summation of the elements of x
SUMF x	A scalar which is the sum of the elements of x
SHIFT (a, x)	An array the same size as x with the elements of x shifted the number of places and in the direction specified by a
INTERVALS x	A scalar which is the number of elements of x minus one
m & n	An array with the number of elements equal to the sum of the number of elements in m and n. The resultant array contains the element(s) of m followed by the element(s) of n

[1] The arguments a, b, and c denote scalar constants, variables, or expressions; x denotes an array variable or expression; m and n denote scalars or arrays.

- EQ - equal
- NE - not equal
- LE - less than or equal
- GE - greater than or equal

can only be used in an IF test. The relation appears between its two arguments which must be scalar constants, variables, or expressions.

2.1.1.7 Order of Computation - Within arithmetic expressions, the order of computation is determined by the relative priorities of the operations -- the operation of highest priority being done first. The priority of operations is as follows:

- Operations within parentheses
- Functions (such as user defined functions, SIN, ARRAY, etc.)
- Exponentiation and negation
- Multiplication and division
- Addition and subtraction
- Relational operations
- Concatenation.

Operations of equal priority are performed sequentially from left to right through an expression, except in the case of exponentiation and negation which are performed from right to left.

2.1.2 Assignment Statement

The assignment statement or substatement is of the form

Variable = Arithmetic Expression .

The variable may be subscripted to assign a value to a single element or a set of elements or to assign values to a subarray. When not subscripted, the variable will automatically assume the dimension of the result of the computation specified by the arithmetic expression.

2.1.3 Program Logic Control Statements

2.1.3.1 The IF Statement - The IF statement is used to control execution based on the relationship between two scalar constants, variables, and/or expressions and is of the form

IF q_1 related to q_2 THEN substatement(s) ELSE substatement(s).

When the relation is satisfied, the THEN clause is executed and the ELSE clause is skipped. When the relation is not satisfied, the THEN clause is skipped and the ELSE clause is executed. The ELSE clause may be omitted from the IF statement, in which case, if the relation fails, the next sequential statement is executed.

An IF test may appear as a substatement. When it is not the last substatement, it must be terminated by the special character ; .

An additional IF statement may appear as a substatement in the THEN or ELSE clause. If it is not the last substatement in the clause, it must be terminated by a semicolon. If the ELSE clause of a nested IF test is to be omitted, it must be replaced by a semicolon.

2.1.3.2 The REPEAT Statement - The REPEAT statement provides iterative execution of a substatement or substatements. It is of the form

REPEAT n, substatement(s)

where n is a positive valued scalar constant, variable, or expression. The substatements are repeated n times (n is rounded to an integer value). REPEAT cannot be used as a substatement.

2.1.3.3 The GO TO Statement - The GO TO statement is used to transfer control to a specified statement. The transfer statement takes the form

GO TO c

where c is an integer or decimal constant denoting a statement number in the program.

2.1.3.4 The EXIT Statement - The EXIT operator forms a complete substatement and is used to create multiple exit points from a user program. EXIT cannot be entered in the conversational mode.

2.1.4 Program Call Statement

A user program is called for execution by entering the program name followed by the parameter string required by the program. When the program execution has been completed, execution will begin in the calling program at the next sequential statement or substatement.

2.1.5 Input/Output Statements

The input and output statements allow the input of data and user programs and the output of data, user text, programs, and system text. Only the statements to input and output data and to output user defined text are allowed as substatements. The form and a description of each input/output statement appears in Table 2-2.

The selection of input and output device and of fixed or floating point format is done using the console sense switches. The input device for the INPUT statement is selected using sense switch 15 as follows:

Switch 15 OFF (down) - console keyboard
Switch 15 ON (up) - card reader.

For all of the output statements, the output device is selected using sense switch 0 as follows:

Switch 0 OFF (down) - typewriter
Switch 0 ON (up) - printer .

For the TYPE, TAB, and PUNCH statements, format selection is controlled by sense switch 1 as follows:

Switch 1 OFF (down) - fixed point
Switch 1 ON (up) - floating point.

TABLE 2-2. INPUT/OUTPUT STATEMENTS

Statement	Description
INPUT variable name	A string of numeric constants in free format is read into the system and associated with the variable named. The variable assumes the dimension of the input string. The numbers may be either integers or decimals in either fixed or floating point format and are separated by commas or blanks. The input string is terminated by two consecutive slashes (//).
TYPE m or TYPE (m, ..., q)[1]	The value(s) of the argument or arguments are printed.
TAB m or TAB (m, ..., q)[1]	The value(s) of the argument or arguments are printed in tabular form.
PUNCH m or PUNCH (m, ..., q)[1]	The value(s) of the argument or arguments are punched on cards and printed.
TYPEOUT 'alphanumeric characters'	The alphanumeric information enclosed in primes is printed.
LIST ALL	The names of all user programs defined in the system are printed.
LIST program name	The source statements of the specified user program are output.
CARD	The source statements of a user program are read into the system from cards.
EXPLAIN ALL	The AMTRAN reserved labels are listed.
EXPLAIN label	An explanation of the specified reserved label is printed.

[1] The arguments m, ..., q may be constants, variables, or expressions.

2.1.6 System Control and Utility Statements

Table 2-3 describes the system control and utility operations available in AMTRAN. The system control labels and any associated arguments cannot be used as substatements.

2.2 DESIGN CONCEPTS

A design concept basic to the AMTRAN software is the execution of an internal form for a user statement rather than the direct execution of the source statement. The translation portion of the system transforms the source statement into the internal form and performs syntax checks. The execution portion of the system interprets the internal form, executes the specified operations, and detects execution errors.

The internal form of a statement consists of a sequence of interpreter instructions which are patterned after a simple assembly language and are executed in order. Each instruction is one or more words, each containing an operator code in the first 7 bits and an operand code in the remaining 9 bits. Many of the instructions implicitly refer to a system pseudo accumulator in which all computation is performed and which varies in length depending on the operator and the operand or operands. Most of the system operations require only one instruction; however, some operators require or may have more than one operand. Additional operands are provided in subsequent words after the operator word. These subsequent words contain a zero value operator to indicate a multiple word instruction. Table 2-4 provides a list of the interpreter operators and the format or formats for each instruction.

With few exceptions, the operand is a three-digit number: the leftmost digit indicates either a program, a constant, or a variable; and the remaining two digits specify a particular operand in the indicated class. The operand is a reference number which the execution

TABLE 2-3. SYSTEM CONTROL AND UTILITY COMMANDS

Command or Symbol	Response
RESET	The system is reinitialized in the conversational mode. All variables are deleted and statement numbers begin at 1.
SUPPRESS	The system enters the SUPPRESS mode where statements are entered without execution. When SUPPRESS is entered in the conversational mode, the current program under construction is saved. When the command is entered in the SUPPRESS or EDIT mode, only the previously saved conversational mode program is maintained.
EDIT	The system reprocesses the user program referenced after the EDIT command. The system requests entry of those statements specified by number in the EDIT statement. The current mode program is saved by the system. The original form of the edited program is maintained on file until the editing process has been completed.
END	When END is entered in the conversational mode, the system is reinitialized in the conversational mode with statement numbers beginning at 1 and all variables retained. When END is entered in the SUPPRESS mode, the current program under construction is destroyed and the previously saved conversational mode program is restored for further statement entry. When END is entered in the EDIT mode, the editing process is terminated and the original unedited program is maintained on file. The system restores the last previous mode program for further statement entry.
DELETE	The system deletes the specified variable(s) and/or console program(s). DELETE may be used only in the conversational mode. If any variables are deleted, statement numbers begin at 1.

TABLE 2-3 - Concluded

Command or Symbol	Response
SAVE	The system performs an END operation, retaining only the specified variables. SAVE may be entered only in the conversational mode.
NAME	The current sequence of statements is stored as a program under the name specified in the NAME statement. Both the source statement and the internal form of the program are saved on disk. All input and output parameters to the program must be declared in the NAME statement. The parameter string is enclosed in parentheses and the individual variable names are separated by commas. The program is a functional program if the program name is also the name of a variable defined in the program (see Section 2.1.1.4).
PAUSE	Execution of the current sequence of operations stops until PROGRAM START is pressed on the console keyboard.
$	The statement line currently being entered is deleted.
$$	The program statement currently being entered is deleted.
#	The preceding character in the statement is deleted.

TABLE 2-4. INTERPRETER INSTRUCTIONS

Operator Code	Operation	Number of Instruction Words	Operand(s)
1	Exit from user program	1	Not used
2	Call user program	1 or more	First word - program Remaining words - variables in the order to be passed as parameters
3	PAUSE	1	Not used
4	INPUT	4	First word - variable Remaining words - variable name (2 characters per word)
5	TYPE	1 or more	Constant(s), variable(s)
6	PUNCH	1 or more	Constant(s), variable(s)
7	TAB	1 or more	Constant(s), variable(s)
9	TYPEOUT	2 or more	First word - number of following words to ouput Remaining words - message (2 characters per word)
12	GO TO	1	244 plus the displacement from the current location to which the branch is to be made
13	LT	1	See GO TO
14	GT	1	See GO TO
15	EQ	1	See GO TO
16	NE	1	See GO TO
17	GE	1	See GO TO
18	LE	1	See GO TO
22	ARRAY, RANGE	3	Scalar constants, variables
23	SHIFT	2	First word - scalar constant or variable Second word - variable
29	MIN	1	Constant, variable, or system accumulator[1]
30	MAX	1	See MIN
31	INTERVALS	1	See MIN
32	SUMF	1	See MIN

TABLE 2-4 - Continued

Operator Code	Operation	Number of Instruction Words	Operand(s)
34	LN	1	See MIN
35	ATAN	1	See MIN
36	ABS	1	See MIN
37	TANH	1	See MIN
38	SUM	1	See MIN
39	MAGNITUDE	1	See MIN
47	SIN	1	See MIN
48	COS	1	See MIN
49	EXP	1	See MIN
50	SQRT	1	See MIN
51	Negation	1	See MIN
54	Store accumulator	1	Variable
		2	First word - variable Second word - scalar variable or constant specifying subscript
		3	First word - variable Second word - scalar variable or constant specifying beginning subscript Third word - scalar variable or constant specifying ending subscript
55	Load accumulator	1	Constant or variable
		2	See store accumulator
		3	See store accumulator
56	Load accumulator and free temporary	1	Variable
57	Free temporary	1	Variable
		2	First word - variable (last in sequence) Second word - variable (first in sequence

TABLE 2-4 - Concluded

Operator Code	Operation	Number of Instruction Words	Operand(s)
58	Exponentiate	1	Constant or variable
59	Multiply	1	Constant or variable
60	Divide	1	Constant or variable
61	Add	1	Constant or variable
62	Subtract	1	Constant or variable
63	Concatenate	1	Constant or variable

[1] The system accumulator is specified by a zero operand.

portion of the system uses to link to the actual data or program. This linkage is provided by information associated with the interpreter instructions being executed. To maintain consistency, this information is available when executing instructions in the conversational mode and in stored programs. The information is provided by a header preceding the executable interpreter instructions and a linkage area following the instructions. This structure for stored programs and for a program during construction is displayed in Figure 2-1.

When the execution portion of the system processes an operand, the header is used to locate either a variable linkage to the data table (and thus to the data area), a constant, or a program name. All of the location entries in the header are relative to the first word of the header. Thus, each program in the system can be executed by the same mechanism and can vary in length, requiring only a minimum amount of storage. This structure provides relocatability for programs. Branches within a program are accomplished by displacements and are therefore relocatable.

The limited internal stoage requires program modularization of the system. The modularity of the system subroutines and the relocatable user program structure imposes an organization on the communication and program area of the system. Figure 2-2 shows the internal storage organization of the system. The majority of COMMON is one integer array in which programs are constructed and executed. To take advantage of the FORTRAN capability of equivalencing floating point and integer arrays and to avoid building separate mechanisms for performing operations on constants and on arrays, the data area and system constants are also included in this one array. All references to data and programs are done through linkage pointers; thus, data elements as well as programs are relocatable within this array. Because user programs are stored on disk and only brought into core when called for execution, the active area is not always required. To minimize core requirements, this area is used during translation for a work area and for tables which are required only during translation.

Program header*	Seven words ● Location of first variable linkage** ● Location of first constant** ● Location of first console program name** ● Number of parameters ● Reference number of program in active table*** ● Reference number of calling program in active table*** ● Location of instruction in calling program to be executed upon return†
Executable interpreter instructions	One word per instruction - not to exceed 245 instructions
Variable linkage area	One word per variable - not to exceed 50 entries
User constants	Two words (1st in odd word) for each distinct (absolute value) constant used by program excluding the system constants - not to exceed 54 constants
Program call table	Four words for each program called - not to exceed 10 programs ● Three words for program name ● One word for reference to active table ***

*The program header must always begin in odd word when in core
**Relative to first word of header
***Determined by row entered in active table when program is executed
†Relative to first work of calling program header

FIGURE 2-1. INTERNAL FORMAT FOR USER PROGRAMS

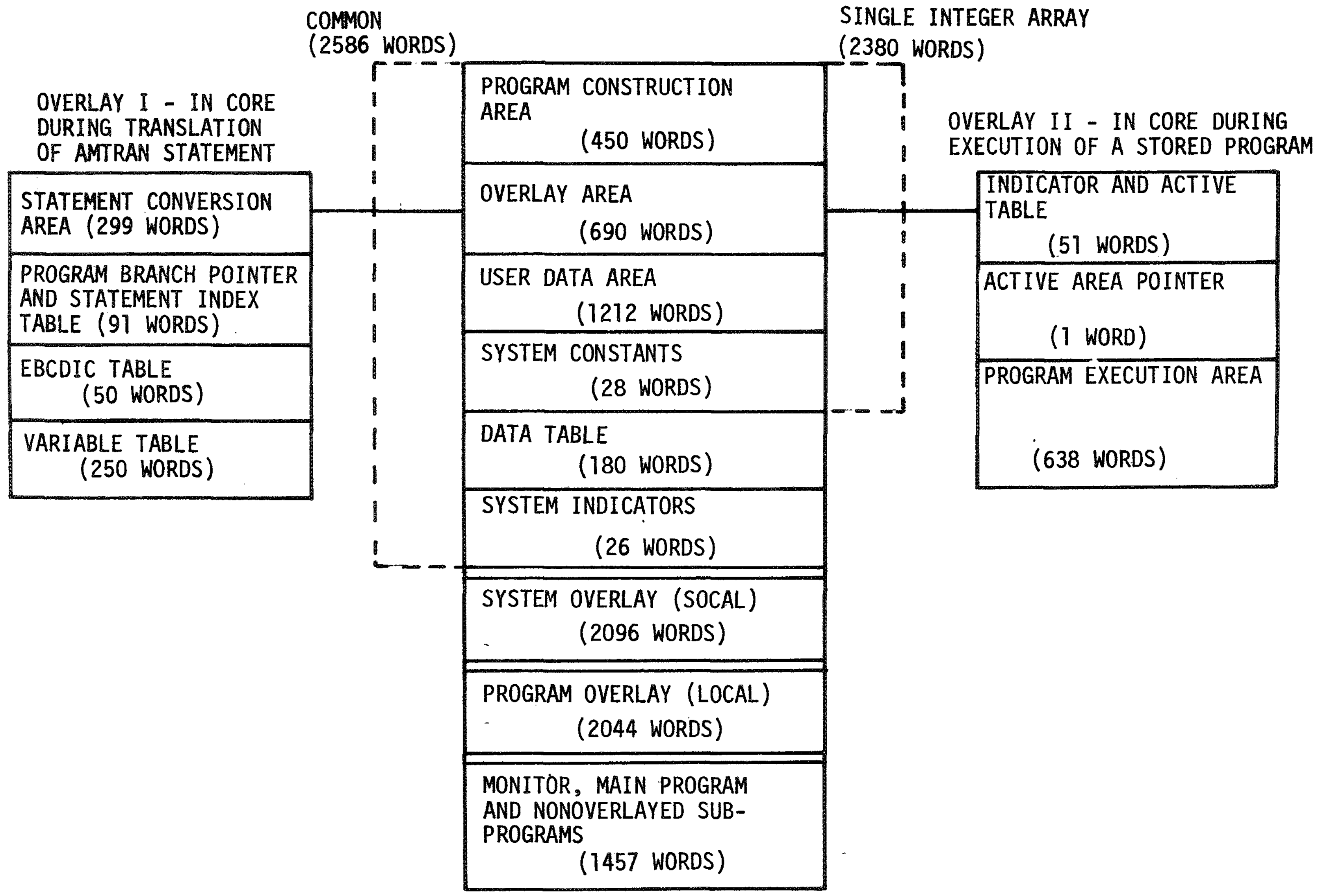

COMMON
(2586 WORDS)
SINGLE INTEGER ARRAY
(2380 WORDS)
OVERLAY I - IN CORE DURING TRANSLATION OF AMTRAN STATEMENT
STATEMENT CONVERSION AREA (299 WORDS)
PROGRAM BRANCH POINTER AND STATEMENT INDEX TABLE (91 WORDS)
EBCDIC TABLE (50 WORDS)
VARIABLE TABLE (250 WORDS)
PROGRAM CONSTRUCTION AREA (450 WORDS)
OVERLAY AREA (690 WORDS)
USER DATA AREA (1212 WORDS)
SYSTEM CONSTANTS (28 WORDS)
DATA TABLE (180 WORDS)
SYSTEM INDICATORS (26 WORDS)
SYSTEM OVERLAY (SOCAL) (2096 WORDS)
PROGRAM OVERLAY (LOCAL) (2044 WORDS)
MONITOR, MAIN PROGRAM AND NONOVERLAYED SUB-PROGRAMS (1457 WORDS)
OVERLAY II - IN CORE DURING EXECUTION OF A STORED PROGRAM
INDICATOR AND ACTIVE TABLE (51 WORDS)
ACTIVE AREA POINTER (1 WORD)
PROGRAM EXECUTION AREA (638 WORDS)

FIGURE 2-2. INTERNAL STORAGE ORGANIZATION

2.3 FUNCTIONAL DESCRIPTION

The software system can be divided into six functional subsystems or programs. Figure 2-3 shows the six functional units and the system flow between them. The following discussions parallel the flow chart and provide a general description of each functional subdivision.

2.3.1 Program ITZ

The program ITZ initializes the system and is executed only once. This program ensures that the user program file is packed. If unpacked, the file is rewritten to remove any unused records between programs. A table, system constants, and several indicators in COMMON and the system working file are initialized.

2.3.2 Subsystem 1

The major task performed by this subsystem is the initialization of the system to accept a new user program. This requires initialization of the program construction area and indicators, the statement index and variable tables, and the user data area. If a change of statement entry mode is requested, this subsystem saves the current mode program or restores a previous mode program as required. This subsystem also stores or deletes programs from the user program file and deletes specified variables from the user data area. The tasks of this subsystem are performed upon initial entry into the system and in response to the following AMTRAN system control commands: RESET, SUPPRESS, END, DELETE, SAVE, NAME, EDIT, and CARD.

2.3.3 Subsystem 2

This subsystem reads an AMTRAN source statement from the selected input device and converts the statement to an internal form which simplifies later translation to executable interpreter instructions. The source statement is saved on the system working file. The source statement in the form of a string of character codes is then scanned

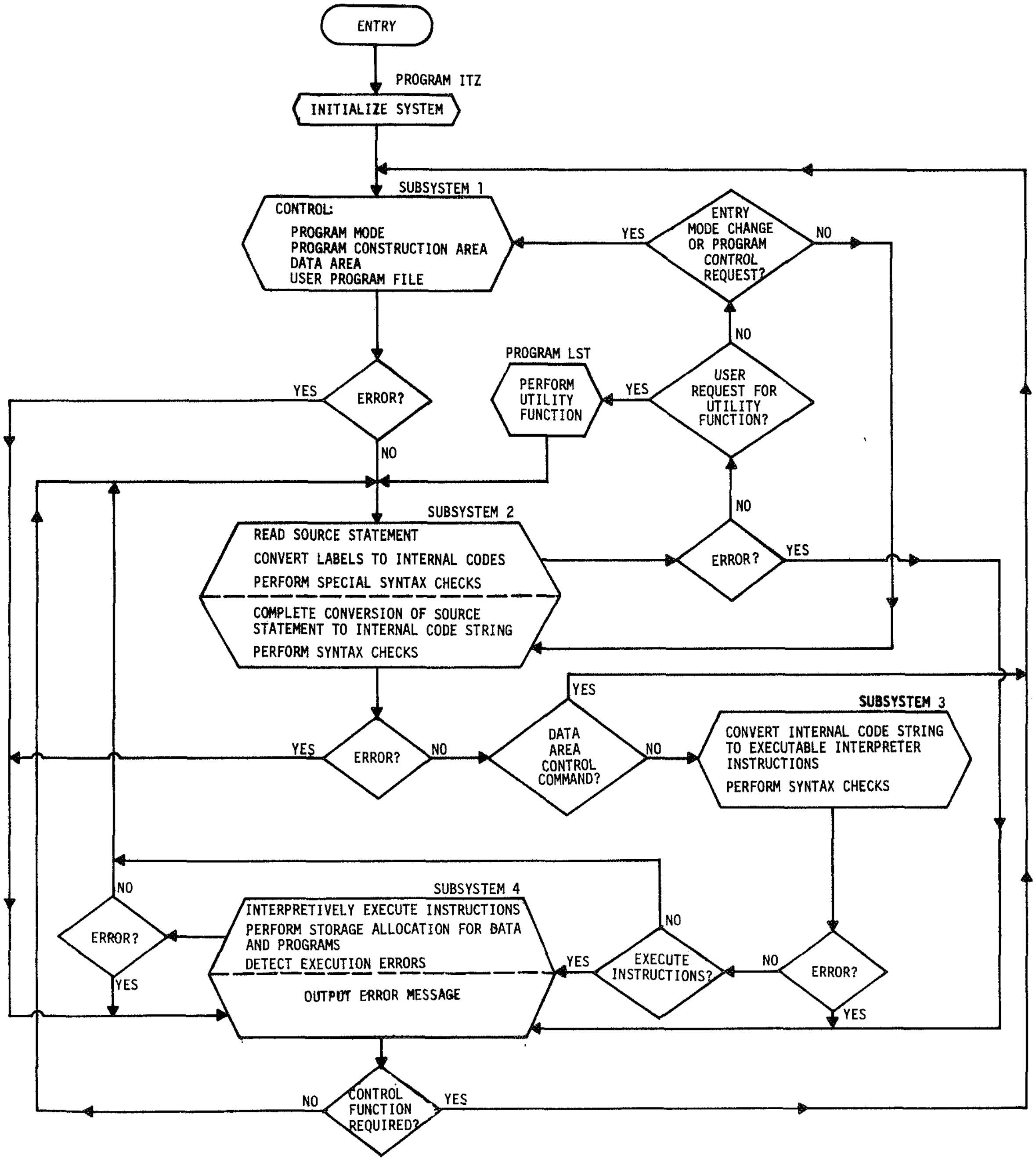

ENTRY
PROGRAM ITZ
INITIALIZE SYSTEM
SUBSYSTEM 1
CONTROL:
PROGRAM MODE
PROGRAM CONSTRUCTION AREA
DATA AREA
USER PROGRAM FILE
YES
ENTRY MODE CHANGE OR PROGRAM CONTROL REQUEST?
NO
NO
PROGRAM LST
PERFORM UTILITY FUNCTION
YES
USER REQUEST FOR UTILITY FUNCTION?
YES
ERROR?
NO
NO
SUBSYSTEM 2
READ SOURCE STATEMENT
CONVERT LABELS TO INTERNAL CODES
PERFORM SPECIAL SYNTAX CHECKS
COMPLETE CONVERSION OF SOURCE STATEMENT TO INTERNAL CODE STRING
PERFORM SYNTAX CHECKS
ERROR?
YES
YES
SUBSYSTEM 3
YES
ERROR?
NO
DATA AREA CONTROL COMMAND?
NO
CONVERT INTERNAL CODE STRING TO EXECUTABLE INTERPRETER INSTRUCTIONS
PERFORM SYNTAX CHECKS
NO
SUBSYSTEM 4
INTERPRETIVELY EXECUTE INSTRUCTIONS
PERFORM STORAGE ALLOCATION FOR DATA AND PROGRAMS
DETECT EXECUTION ERRORS
OUTPUT ERROR MESSAGE
ERROR?
YES
NO
YES
EXECUTE INSTRUCTIONS?
NO
ERROR?
YES
NO
CONTROL FUNCTION REQUIRED?
YES

FIGURE 2-3. SYSTEM FUNCTIONAL FLOW CHART

twice to convert it to a compressed internal code string and to perform syntax checks. The system control and utility commands RESET, SUPPRESS, END, NAME, CARD, LIST, and EXPLAIN are detected during the first scan and the system flow is routed to subsystem 1 or program LST as required. During the second scan, the system control commands DELETE and SAVE are detected and system flow is routed to subsystem 1.

2.3.4 Subsystem 3

This subsystem converts the internal code string generated by subsystem 2 to executable interpreter instructions which are to be executed in order by subsystem 4. The interpreter instructions are described in Table 2-4. Syntax checks are performed throughout the subsystem.

2.3.5 Subsystem 4

This subsystem executes the instructions generated by subsystem 3. For each instruction, the operand (or operands) is located, validity checks are performed, and the instruction is executed. The subsystem performs dynamic storage allocation for data and user programs and maintains the linkage between user programs in the execution string. When an error is detected, the subsystem terminates execution, follows the established program linkage back to the conversational mode program, releases any data associated with called user programs, and prints the required error message. Error messages requested by other subsystems or programs are also output by this subsystem.

2.3.6 Program LST

This program, in response to user requests, performs the output of the names of all user programs defined in the system, the source statements of a user program, the system reserved labels, and prestored explanations of system reserved labels.

2.4 SYSTEM DATA BASE

The system data base consists of FORTRAN COMMON and three files on disk. For ease in referencing the data base in subsystem and program documentation, each item in the system data base has been assigned a reference number. A deviation from the standard numbering system has been used for items which have several uses. A dash followed by a number is used to distinguish between various uses of an item. Throughout the documentation, reference numbers are enclosed in brackets, { }.

2.4.1 FORTRAN COMMON

COMMON consists of nineteen integer variables: KODE (2380), IDAT (90, 2), IT(10), I, J, L, II, ID, IE, IFT, KON, LNT, LPV, NCP, NV, IEX, IGT, NMB, and NAP. The use of each variable is described in the following sections.

2.4.1.1 KODE (2380) {1.} - The major activities of the system are performed using this array. It contains the program construction area, system tables, user program execution area, and the user data area. The substructure of one- and two-dimensional arrays within KODE is described below. The subscript and dimension specify, respectively, the beginning subscript and the size of each array in KODE.

Reference Number	Subscript	Dimension	Contents
{1.1}	1	450	Program construction area. The program construction area has the same structure as a stored user program with the exception of the 4th through 7th words of the header. See Figure 2-1 for the program structure.
	1	1	252

Reference Number	Subscript	Dimension	Contents
	2	1	302
	3	1	410
{1.1.1}	4	1	User program count - the current number of user programs referenced by the user program being constructed, i.e., the number of entries in the program call table {1.1.8}; may range from 0 through 10.
{1.1.2}	5	1	Previous program count - the number of user programs referenced after the last user program statement; may range from 0 through 10.
{1.1.3}	6	1	Previous variable count - the number of variables referenced after the last statement, i.e., the previous number of entries in the variable table {1.2-1.5}; may range from 0 through 29.
{1.1.4}	7	1	Card status: 1 - program not entered from cards 2 - program originally entered on cards and stored on the system working file {20.} 3 - initiate read of program from cards.
{1.1.5}	8	245	Interpreter instructions area - instructions to be executed in the order of appearance, may contain 0 to 245 instructions. Instructions start at KODE(8) and are contiguous. See Table 2-4 for the interpreter instructions.

Reference Number	Subscript	Dimension	Contents
{1.1.6}	253	50	Variable linkage area - each nonzero entry points to the row in the data table {2.} containing the storage information for the variable. The 50 positions correspond to the 50 rows in the variable table {1.2-1.5}.
{1.1.7}	303	2, 54	User constants area - contains each constant entered by the user in the program under construction; the constants are stored in absolute value floating point.
{1.1.8}	411	4, 10	Program call table - contains one entry for each user program called; entries are in contiguous columns beginning with column 1 Rows 1 - 3: console program name (2 characters per word) Row 4: last active reference number, i. e., row in active table {1.2-2.1} where the name was entered when last called for execution by the program being constructed.
{1.2}	451	690	Overlay area for system tables and user programs
{1.2-1}			Overlay I - system tables for translation of AMTRAN statements
{1.2-1.1}	451	299	Statement conversion area - used for translation of a source statement. The statement is passed between programs in one of four states (for each state, programs use the remainder of the area for working storage.)

Reference Number	Subscript	Dimensions	Contents
{1.2-1.1} (Continued)			State 1 - statement begins at subscript 511 and can be up to 218 words long State 2 - statement ends at subscript 749 and may begin at no lower than subscript 511 State 3 - statement begins at subscript 461 and may be up to 289 words long State 4 - statement begins at subscript 451 and may be up to 100 words long.
{1.2-1.2}	750	1	Program branch pointer - points to the statement entry in the statement index table {1.2-1.3} for which the corresponding interpreter instructions begin at or beyond subscript 129 in the program construction area {1.1}.
{1.2-1.3}	45, 2	1	Statement index table - contains one entry for each statement entered in the program being constructed; entries are in contiguous rows beginning with row 1 Column 1: statement number, Leftmost 8 bits - 2-digit integer to left of decimal point Rightmost 8 bits - 2-digit integer to right of decimal point Column 2: statement length or statement length and location,

Reference Number	Subscript	Dimension	Contents
{1.2-1.3} (Continued)			Form 1 - number of records on the system working file {20.} for the AMTRAN source statement Form 2 - Leftmost 9 bits - number of records on the system working file {20.} for the source statement Rightmost 9 bits - subscript in program construction area {1.1} where interpreter instructions begin for this statement. If the row in the table is less than the program branch pointer {1.2-1.2}, this contains the actual subscript. If the row is greater than or equal to the program branch pointer, this contains the actual subscript minus 127.
{1.2-1.4}	841	50	EBCDIC table - EBCDIC codes for the AMTRAN characters ordered by the AMTRAN character codes. The EBCDIC code for each character appears in the left half of the word with an EBCDIC coded blank in the right half of the word.
{1.2-1.5}	891	50, 5	Variable table - Rows 1 - 29: one entry for each variable referenced in the program under construction; entries are contiguous and made in the order of first appearance beginning with row 1,

Reference Number	Subscript	Dimension	Contents
{1.2-1.5} (Continued)			Columns 1 - 3: variable name (2 characters per word) Column 4: 0 - variable undefined 1 - variable defined (has appeared to the left of an =) Column 5: 1 - column 4 cannot be changed 0 - initial value -2 - column 4 can be changed Rows 30 - 50: the following areas are used, the remainder of the table is unused. Row 30, Column 4: indicates the requirement of a temporary variable for the count in a REPEAT statement 0 - not required 1 - required Rows 30 - 50, Column 4: one word for each possible temporary variable 0 - variable not required 1 - variable required for storage of temporary results. Rows 32 - 42, Columns 2, 3: control tables required by the program SCB.

Reference Number	Subscript	Dimension	Contents
{1.2-2}			Overlay II - used for execution of user programs when called.
	451	1	1
{1.2-2.1}	452	10, 5	Active table - contains one entry for each user program in the program execution area {1.2-2.3}; entries are not necessarily contiguous, Column 1: subscript in KODE at which program begins (location of the first word of the program header) Column 2: length of the program + - program currently in execution chain - - program not in execution chain Columns 3 - 5: program name (2 characters per word).
{1.2-2.2}	502	1	Active area pointer - points to subscript where next program may begin in the program execution area {1.2-2.3}.
{1.2-2.3}	503	638	Program execution area.
{1.3}	1141	1240	Data area - data is stored in floating point and accessed for arithmetic operations by equivalencing a floating point array to KODE. Data is stored beginning in an odd subscripted location.

<table>
<tr><th>Reference Number</th><th>Subscript</th><th>Dimension</th><th>Contents</th></tr>
<tr><td>{1.3.1}</td><td>1141</td><td>1212</td><td>User data area - storage for user variables, temporary variables, and the system accumulator - initial values are assigned to the following:
<table>
<tr><th>Subscript</th><th>Value</th></tr>
<tr><td>1141</td><td>1145</td></tr>
<tr><td>1142</td><td>604</td></tr>
<tr><td>1145</td><td>0</td></tr>
<tr><td>1146</td><td>0.</td></tr>
</table></td></tr>
<tr><td>{1.3.2}</td><td>2353</td><td>28</td><td>System constants - contains the following floating point numbers in the order listed: 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 3.1415927, 57.2958, 0.0174533.</td></tr>
</table>

2.4.1.2 IDAT (90,2)' {2.} - Data table used to link the variable linkage area {1.1.6} to data in the user data area {1.3.1}. Contains one entry for each data set. Entries are in contiguous rows beginning with row 1.

Column 1 - floating point subscript of first element in data set

Column 2 - number of elements in data set.

Note: Row 90 is reserved for the system accumulator.

2.4.1.3 IT (10){3.} - General working array. The ten entries have the varying uses described below.

Reference Number	Subscript	Contents
{3.1-1}	1	During EDIT mode - location in user program table {21.4} of program being edited.

Reference Number	Subscript	Contents
{3.1-2}		During interpreter execution - the interpreter operator being executed.
{3.2-1}	2	During EDIT mode - source statement record pointer - record number on the user program file {21.} where the next source statement begins.
{3.2-2}		During interpreter execution - operand or first operand for operator being executed {3.1-2}.
{3.3}	3	During EDIT - points to entry in statement index table {1.2-1.3} for last statement edited.
{3.4}	4	Type of entry to program RST: 1 - entry for normal initialization of program construction area 2 - entry from program DLT, indicating a program has been deleted from the user program file {21.} 3 - entry from program NAM, indicating a program has been stored on the user program file 4 - entry from subsystem 2, indicating a request for a change of statement entry mode 5 - entry from program DLT, indicating data storage has been made available.
{3.5}	5	Unused.
{3.6}	6	Source statement record count - one plus the number of records occupied on the system working file {20.} by the program source statements, i.e., the relative record for the storage of the next source statement. The record number is relative to a base record determined by the statement entry mode {16.} - may be in file area {20.1}, {20.2}, or {20.3}.

Reference Number	Subscript	Contents
{3.7-1}	7	Type of entry to program DLT: 1 - perform system control commands SAVE or DELETE 2 - perform system control commands SAVE ALL or END (when entered in the conversational mode).
{3.7-2}		Subclass indicator for programs LSG and TRG.
{3.7-3}		Type of entry to program RTN: 1 - normal return from user program 2 - return from user program and output error message 3 - output error message.
{3.8}	8	Overlay status: 1 - system tables in core - Overlay I {1.2-1} 2 - user programs in core - Overlay II {1.2-2}.
{3.9-1}	9	When processing the EXPLAIN operator - number indicating which explanation to output from the system control file {22.7}.
{3.9-2}		When processing the LIST program operator - record on the user program file {21.} where internal form of program begins.
{3.9-3}		During interpreter execution - storage allocation parameter.
{3.10-1}	10	Type of entry to program LST: - - process LIST ALL operator 0 - process EXPLAIN operator + - process LIST program operator - number of words in source program to be listed. Program is stored 2 characters per word.
{3.10-2}		During interpreter execution - storage allocation parameter.

2.4.1.4 I{4.} - General working register with the following special uses:

Reference Number	Use
{4-1}	During interpreter execution - current location - subscript location in KODE of interpreter instruction which is currently being executed.
{4-2}	On entry to program EDT - displacement relative to KODE (510) at which scan for statement numbers is to begin
{4-3}	On entry to program SCB - displacement relative to KODE (510) at which scan is to begin.

2.4.1.5 J {5.} - During interpreter execution - operator class.

2.4.1.6 L {6.} - Last instruction pointer - subscript in interpreter instructions area {1.1.5.} of last interpreter instruction to be executed, i.e., the last instruction generated for the current statement.

2.4.1.7 II {7.} - FORTRAN record control for all files.

2.4.1.8 ID {8.} - Current program pointer - subscript of location in KODE containing the first word of the header for the user program currently being executed.

2.4.1.9 IE {9.} - Error indicator:

0: no error
>0: reference number for error.

2.4.1.10 IFT {10.} - Data table entry count - number of entries in the data table {2.} excluding the entry (row 90) for the system accumulator; may range from 0 through 89.

2.4.1.11 KON {11.} - Constant count - current number of user constants entered in the program being constructed, i.e., the number of entries in the user constants area {1.1.7}; may range from 0 through 54.

2.4.1.12 LNT {12.} - Data storage count - number of floating point words currently available in the user data storage area {1.3.1}; may range from 0 through 604.

2.4.1.13 LPV {13.} - Last previous instruction pointer - subscript of location in the interpreter instructions area {1.1.5} containing the last interpreter instruction generated for the previous program statement.

2.4.1.14 NCP {14.} - Number of user programs stored on the user program file {21.}; same value as {21.3}.

2.4.1.15 NV {15.} - Variable count - current number of variables referenced by the program being constructed, i.e., the number of variable names entered in the variable table {1.2-1.5}.

2.4.1.16 IEX {16.} - Statement entry mode:

1 - conversational mode, statements executed when entered

2 - SUPPRESS mode, statements not executed (programs entered on cards are processed in the SUPPRESS mode)

3 - EDIT mode, program reprocessed with alterations but without execution.

2.4.1.17 IGT {17.} - Subscript of location in KODE at which an interpreter branch instruction was last executed.

2.4.1.18 NMB {18.} - Statement count - number of statements entered, i.e., points to the entry in statement index table {1.2-1.3} for the current statement.

2.4.1.19 NAP {19.} - Active program count - number of entries in the active table {1.2-2.1}, i.e., number of programs in the program execution area {1.2-2.3}.

2.4.2 Disk Files

2.4.2.1 System Working File {20.} - The system working file consists of 7,360 one-word records used for the following information:

Reference Number	Beginning Record Number	Number of Records	Contents
{20.1}	1	1140	Source statements for conversational mode program
{20.2}	1141	1140	Source statements for SUPPRESS mode program
{20.3}	2281	1140	Source statements for EDIT mode program
{20.4}	3421	1170	Conversational mode program and indicators saved during SUPPRESS mode
{20.5-1}	4591	1170	Last previous mode program and indicators saved during EDIT mode
{20.5-2}	4591	1600	Temporary storage of first 1,600 words of KODE when area is needed for repacking of the user program file by the program RST
{20.6}	6191	400	Overlay I {1.2-1}, except statement conversion area {1.2-1.1}
{20.7}	6591	391	Overlay II {1.2-2}
	6982	379	Unused.

2.4.2.2 User Program File {21.} - The user program file consists of 30,720 one-word records organized in the following manner:

Reference Number	Beginning Record Number	Number of Records	Contents
{21.1}	1	1	File status: 0 - packed 1 - not packed
{21.2}	2	1	First record available for program storage
{21.3}	3	1	Number of user programs stored on file (Same value as {14. }.)
{21.4}	4	96, 6	User program table - one entry for each program; entries arranged in alphabetical order and in contiguous rows beginning in row 1. Columns 1 - 3: program name (2 characters per word) Column 4: record number at which internal form of program begins (see Figure 2-1 for program structure) Column 5: record number at which the source statement count is stored. This is followed by the statement index table and the source statements of the program. Column 6: number of records occupied by statement count, statement index table, and source statements
{21.5}	580	30, 141	Records for program storage.

2.4.2.3 System Control File {22.} - The system control file consists of 14, 080 one-word records organized in the following units:

Reference Number	Beginning Record Number	Number of Records	Contents
{22.1}	1	56, 4	System label table - Columns 1 - 3: system label (2 characters per word) Column 4: control code for scanning.
{22.2}	401	15	Array containing codes used by program SCB to reformat REPEAT statement
{22.3}	467	34	Initialization array for programs STV and TAB.
{22.4}	501	50	EBCDIC table - EBCDIC codes for AMTRAN character set ordered by AMTRAN character codes (same as {1.2-1.4}).
{22.5}	601	100	Error message control array - one entry for each error message contains the record number at which the message begins.
{22.6}	701	1399	Error messages - stored 2 characters per word.
{22.7}	2100	2, 56	System explanation control table - the order of entries in this table corresponds to the labels entered in the system label table, Row 1: record number at which explanation begins Row 2: number of records in explanation.
{22.8}	2212	11, 869	Explanations for system labels (2 characters per word).

3. SUBSYSTEM DESCRIPTIONS

The four subsystems presented in the functional description (see Section 2.3) and flowcharted in Figure 2-3 are described further in this section. The specific tasks required to achieve the functional capabilities of each subsystem are presented. Flowcharts are provided to show the structure of the programs comprising each subsystem. The flowchart elements appearing in dotted lines are tasks performed for a subsystem by the system control program CTL.

3.1 SUBSYSTEM 1

The major task for this subsystem is the initialization of the system to accept a new user program. The three possible entries to the subsystem are for

- Normal initialization (entry 1)
- Program control commands (entry 2)
- Data area control commands (entry 3).

The various entries and the actions taken are shown in Figures 3-1 and described below.

3.1.1 Normal Initialization

Upon a normal entry to the subsystem, program and data initialization are performed. For program initialization, the following are initialized:

- Program construction area {1.1} (except card status {1.1.4})
- Program branch pointer {1.2-1.2}
- Statement index table {1.2-1.3}
- Variable table {1.2-1.5}
- Source statement record count {3.6}
- Last instruction pointer {6.}
- Error indicator {9.}
- Constant count {11.}
- Last previous instruction pointer {13.}
- Statement count {18.}.

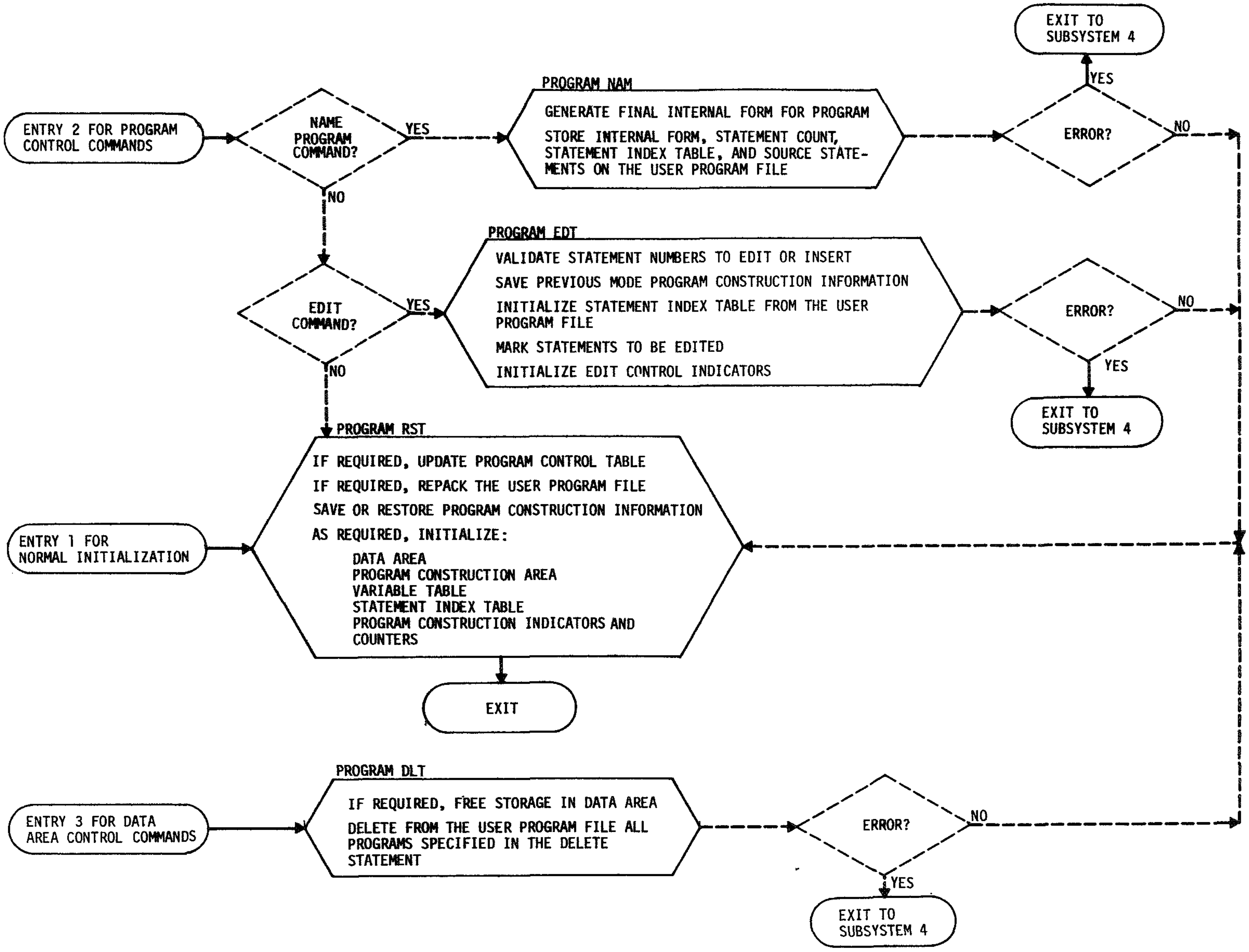

ENTRY 2 FOR PROGRAM CONTROL COMMANDS
NAME PROGRAM COMMAND?
YES
NO
PROGRAM NAM
GENERATE FINAL INTERNAL FORM FOR PROGRAM
STORE INTERNAL FORM, STATEMENT COUNT, STATEMENT INDEX TABLE, AND SOURCE STATEMENTS ON THE USER PROGRAM FILE
ERROR?
YES
EXIT TO SUBSYSTEM 4
NO
EDIT COMMAND?
YES
NO
PROGRAM EDT
VALIDATE STATEMENT NUMBERS TO EDIT OR INSERT
SAVE PREVIOUS MODE PROGRAM CONSTRUCTION INFORMATION
INITIALIZE STATEMENT INDEX TABLE FROM THE USER PROGRAM FILE
MARK STATEMENTS TO BE EDITED
INITIALIZE EDIT CONTROL INDICATORS
ERROR?
NO
YES
EXIT TO SUBSYSTEM 4
ENTRY 1 FOR NORMAL INITIALIZATION
PROGRAM RST
IF REQUIRED, UPDATE PROGRAM CONTROL TABLE
IF REQUIRED, REPACK THE USER PROGRAM FILE
SAVE OR RESTORE PROGRAM CONSTRUCTION INFORMATION
AS REQUIRED, INITIALIZE:
DATA AREA
PROGRAM CONSTRUCTION AREA
VARIABLE TABLE
STATEMENT INDEX TABLE
PROGRAM CONSTRUCTION INDICATORS AND COUNTERS
EXIT
ENTRY 3 FOR DATA AREA CONTROL COMMANDS
PROGRAM DLT
IF REQUIRED, FREE STORAGE IN DATA AREA
DELETE FROM THE USER PROGRAM FILE ALL PROGRAMS SPECIFIED IN THE DELETE STATEMENT
ERROR?
NO
YES
EXIT TO SUBSYSTEM 4

FIGURE 3-1. SUBSYSTEM 1

For data initialization, the following are initialized:

- User data area {1.3.1}
- Data table entry count {10.}
- Data storage count {12.}.

3.1.2 Program Control Commands

The actions taken in response to the program control commands, i.e., SUPPRESS, END, CARD, EDIT, and NAME, are described below.

3.1.2.1 SUPPRESS - If the system is in the EDIT mode when the SUPPRESS command occurs, the system restores the program construction information on the system working file {20.5-1} for the previous mode. If the system is in the conversational mode, the following program construction information is saved on the system working file {20.4}:

- Program construction area {1.1}
- Overlay I {1.2-1}, except the statement conversion area {1.2-1.1}
- All remaining variables in COMMON, except the data table {2.} and the number of user programs {14.}.

In all modes, the statement entry mode {16.} is reset and normal program initialization is done.

3.1.2.2 END - When the END command is entered in the conversational mode, all temporary data storage is freed and normal program initialization is done.

When the END command is entered in the SUPPRESS or EDIT mode, the previously saved program construction information is restored (see Section 3.1.2.1).

3.1.2.3 CARD - The subsystem will process the CARD command as a SUPPRESS command (see Section 3.1.2.1), with the additional action of setting the card status {1.1.4} to three.

3.1.2.4 EDIT - The subsystem will perform ten tasks in response to an EDIT command: validate the statement numbers to be edited or inserted; save the program construction information (see Section 3.1.2.1) on the system working file {20.5-1} when in the conversational or SUPPRESS mode; initialize the statement index table {1.2-1.3} from the table stored on the user program file {21.} with the program to be edited; mark the statements to be edited and/or create entries for statements to be inserted; initialize the source statement record count {3.2-1} and the pointer to the last statement edited {3.3}; set the statement entry mode {16.} to the EDIT mode; and perform normal program initialization, omitting initialization of the statement index table.

3.1.2.5 NAME - In response to the NAME command, the information in the program construction area {1.1} is processed to generate the final internal form of the program. Validity checks are performed on the program name, parameter string, and interpreter branch instructions. The internal form of the program is stored on the user program file {21.5}, along with the following information:

- Statement count {18.}
- Statement index table {1.2-1.3}, omitting unused rows and placing column two in form 1
- Program source statements from the system working file {20.}.

The program name and storage information are entered in the program control table {21.4} and the record control indicators {21.1}, {21.2}, and {21.3} are updated on the program file. The performance of the remaining tasks is dependent on the statement entry mode: if the program was entered in the conversational mode, perform normal program and data initialization; if the program was entered in the

SUPPRESS mode, restore the previously saved program construction information (see Section 3.1.2.1) for the conversational mode; if the program was entered in the EDIT mode, repack the program file {21.} and restore the program construction information for the last previous mode.

3.1.3 Data Area Control Commands

The response of the subsystem to the data area control commands SAVE and DELETE is explained below.

3.1.3.1 SAVE - For the SAVE ALL command, the subsystem frees all temporary storage in the data area and performs normal program initialization.

In response to the SAVE command followed by variable names, the subsystem frees all elements in the user data area {1.3.1} not specified in the SAVE statement and performs normal program initialization.

3.1.3.2 DELETE - Programs specified in the DELETE statement are deleted from the user program file {21.} and the program file is packed. Variables specified in the DELETE statement and temporary variables are freed in the user data area {1.3.1}. The normal program initialization is performed if any variables are deleted.

3.2 SUBSYSTEM 2

Subsystem 2, shown in Figure 3-2, reads an AMTRAN source statement, converts the statement to an internal code string suitable for translation to executable interpreter instructions, and performs syntax checks.

The AMTRAN source statement is read into the statement conversion area {1.2-1.1} from either the keyboard, card reader, or disk

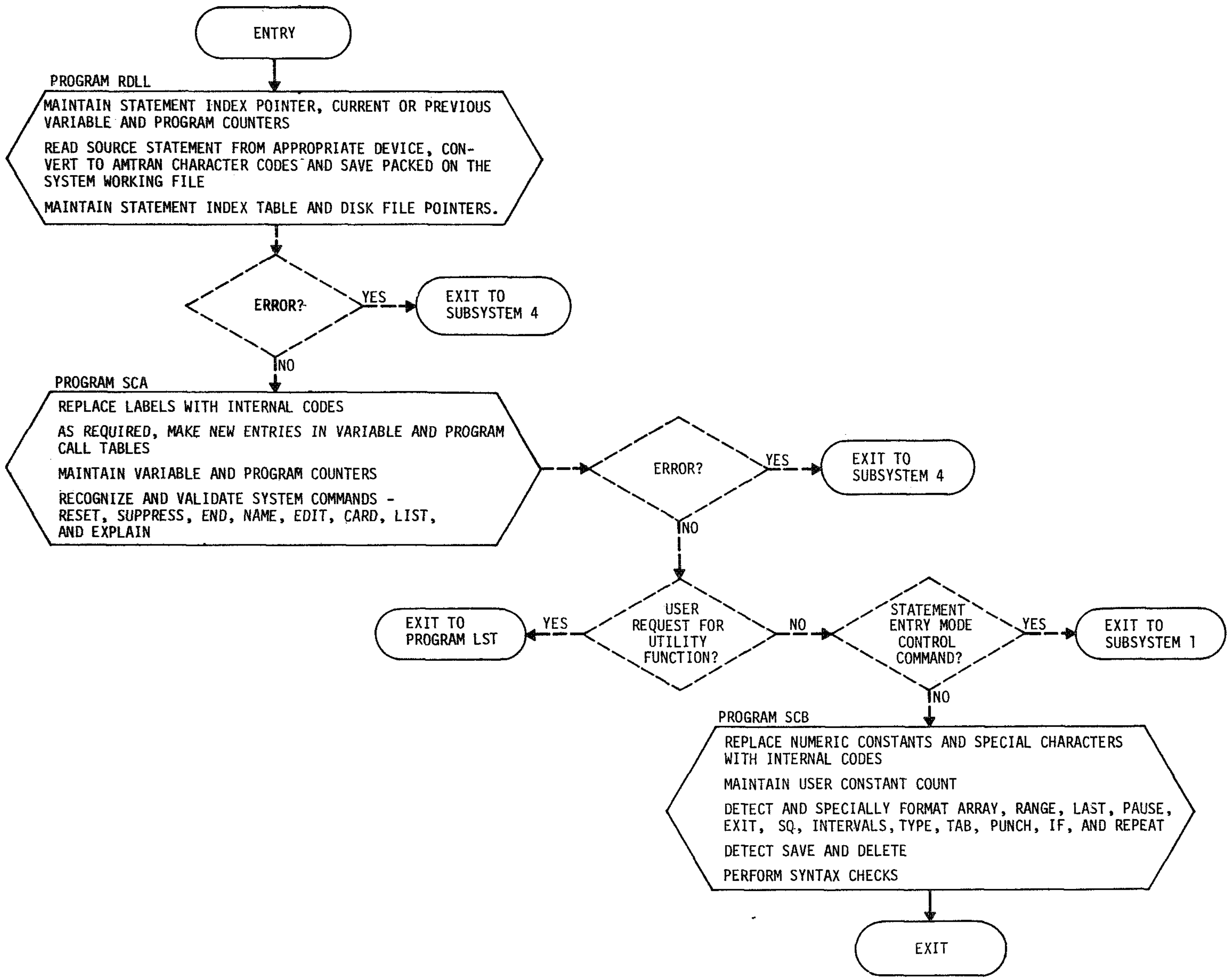

ENTRY
PROGRAM RDLL
MAINTAIN STATEMENT INDEX POINTER, CURRENT OR PREVIOUS VARIABLE AND PROGRAM COUNTERS
READ SOURCE STATEMENT FROM APPROPRIATE DEVICE, CONVERT TO AMTRAN CHARACTER CODES AND SAVE PACKED ON THE SYSTEM WORKING FILE
MAINTAIN STATEMENT INDEX TABLE AND DISK FILE POINTERS.
ERROR?
YES
EXIT TO SUBSYSTEM 4
NO
PROGRAM SCA
REPLACE LABELS WITH INTERNAL CODES
AS REQUIRED, MAKE NEW ENTRIES IN VARIABLE AND PROGRAM CALL TABLES
MAINTAIN VARIABLE AND PROGRAM COUNTERS
RECOGNIZE AND VALIDATE SYSTEM COMMANDS - RESET, SUPPRESS, END, NAME, EDIT, CARD, LIST, AND EXPLAIN
ERROR?
YES
EXIT TO SUBSYSTEM 4
NO
USER REQUEST FOR UTILITY FUNCTION?
YES
EXIT TO PROGRAM LST
NO
STATEMENT ENTRY MODE CONTROL COMMAND?
YES
EXIT TO SUBSYSTEM 1
NO
PROGRAM SCB
REPLACE NUMERIC CONSTANTS AND SPECIAL CHARACTERS WITH INTERNAL CODES
MAINTAIN USER CONSTANT COUNT
DETECT AND SPECIALLY FORMAT ARRAY, RANGE, LAST, PAUSE, EXIT, SQ, INTERVALS, TYPE, TAB, PUNCH, IF, AND REPEAT
DETECT SAVE AND DELETE
PERFORM SYNTAX CHECKS
EXIT

FIGURE 3-2. SUBSYSTEM 2

and is saved on the system working file (either {20.1}, {20.2}, or {20.3}). The source statement is then scanned to convert labels, numeric constants, and special characters to internal codes. As required, the subsystem updates the following:

- Statement count {18.}
- Statement index table {1.2-1.3}
- Variable count {15.}
- Previous variable count {1.1.3}
- Variable table {1.2-1.5}
- Program count {1.1.1}
- Previous program count {1.1.2}
- Program call table {1.1.8}
- Constant count {11.}
- User constants area {1.1.7}
- Source statement record pointer for the user program file {3.2-1}
- Pointer to the last statement edited {3.3}
- Source statement record count for the system working file {3.6}.

Special formatting of the statement string and preliminary syntax checks are done for the operators ARRAY, RANGE, LAST, PAUSE, EXIT, SQ, INTERVALS, TYPE, TAB, PUNCH, and TYPEOUT, and for the IF and REPEAT statements.

The normal exit from the subsystem is to subsystem 3 with the internal code string in state 3 of the statement conversion area {1.2-1.1}. However, alternate exits occur when the subsystem recognizes certain system control and utility commands. The commands and the tasks performed in response by the subsystem are discussed below.

3.2.1 RESET

On a RESET command, the subsystem sets the statement entry mode {16.} to the conversational mode and the previous variable count {1.1.3} to zero, and routes the system flow to subsystem 1, entry 1.

3.2.2 SUPPRESS or CARD

The subsystem sets the type of entry to program RST {3.4} to four and routes the system flow to subsystem 1, entry 2.

3.2.3 END

When END is entered in the conversational mode, the type of entry to program DLT {3.7-1} is set to two and the system flow is routed to subsystem 1, entry 3. In the SUPPRESS or EDIT mode, the subsystem performs the same tasks as for SUPPRESS.

3.2.4 NAME

The system performs syntax checks and formats the program name. The system flow is routed to subsystem 1, entry 2.

3.2.5 EDIT

To process an EDIT command, the subsystem performs syntax checks, initializes the pointer to the program table {3.1-1} and the FORTRAN record control {7.}, and routes the system flow to subsystem 1, entry 2.

3.2.6 LIST or EXPLAIN

For a LIST or EXPLAIN command, the subsystem sets the type of entry to program LST {3.10-1}, the EXPLAIN or LIST control indicator {3.9-1,-2}, and the FORTRAN record control {7.}, and routes the system flow to the program LST.

3.2.7 DELETE or SAVE

On a DELETE or SAVE, system flow is routed to subsystem 1, entry 3.

3.3 SUBSYSTEM 3

Subsystem 3, flowcharted in Figure 3-3, converts the internal code string (state 3 of the statement conversion area {1.2-1.1}) generated by subsystem 2 to the corresponding sequence of interpreter instructions executable by subsystem 4 and performs syntax checks. From the internal code string, the subsystem first generates a post-fix Polish stack based on the operation priorities (see Section 2.1.1.7). The stack is generated so that the order in which operations are to be performed corresponds to their order of occurrence from left to right and so that the associated operands immediately precede an operation. The subsystem then uses this stack to generate the executable interpreter instructions. Moving along the stack from left to right, the instructions to perform each operation and to store the result are generated. The result then replaces the operator and operands in the stack and the stack is compressed. This process is continued until the stack has been transformed into the corresponding interpreter instructions.

The generated instructions are entered into the interpreter instructions area {1.1.5} at the location specified by the value of the pointer to the last interpreter instruction {13.}. For the current statement entry, column 2 of the statement index table {1.2-1.3} is changed to form 2. The current interpreter instruction limit {6.} is set.

3.4 SUBSYSTEM 4

Subsystem 4 executes the interpreter instructions generated by subsystem 3. The instructions to be executed are specified by the pointer to the last instruction generated for the previous statement

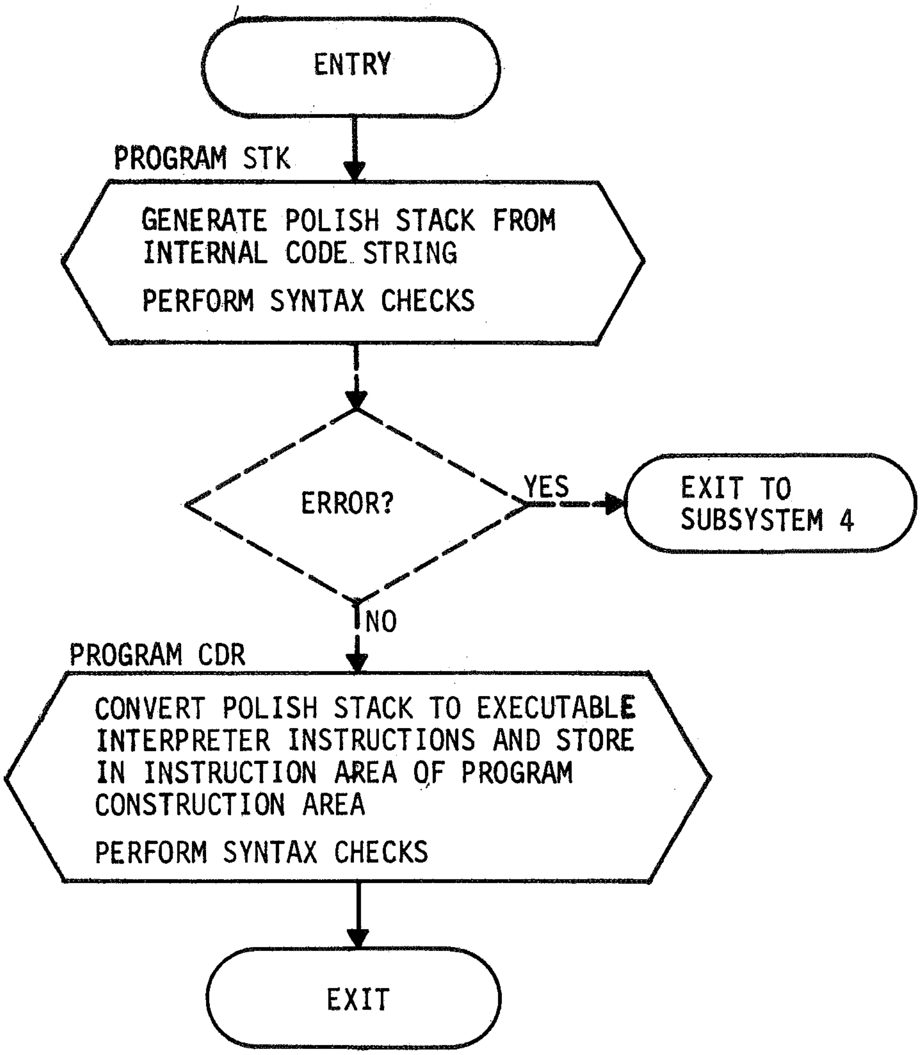

ENTRY
PROGRAM STK
GENERATE POLISH STACK FROM INTERNAL CODE STRING
PERFORM SYNTAX CHECKS
ERROR?
YES
EXIT TO SUBSYSTEM 4
NO
PROGRAM CDR
CONVERT POLISH STACK TO EXECUTABLE INTERPRETER INSTRUCTIONS AND STORE IN INSTRUCTION AREA OF PROGRAM CONSTRUCTION AREA
PERFORM SYNTAX CHECKS
EXIT

FIGURE 3-3. SUBSYSTEM 3

{13.} and the current instruction limit {6.}. The instructions are executed in the order of occurrence. Each instruction is upacked, separating the operator code {3.1-2} and the operand code {3.2-2}. The operator is then classified according to the following:

Class	Subclass	Operators (See Table 2-4 for the Operator Codes)
1		Exit from user program
2		Call user program
3		INPUT
4		TYPE, PUNCH, TYPEOUT
5	1	Load accumulator, load accumulator and free temporary
	2	SIN, COS, EXP, SQRT, NEG
	3	+, *, /, -, &
	4	GO TO, LT, GT, EQ, NE, GE, LE
	5	Store accumulator
	6	Free temporary
6	1	ARRAY or RANGE
	2	MIN, MAX, INTERVALS, SUMF
	3	LN, ATAN, ABS, TANH, SUM, MAGNITUDE
	4	SHIFT
	5	**
7		PAUSE
8		TAB.

The subclass indicator {3.7-2} is set, if required by the classification. Depending on the operator classification number., the system flow is routed to the appropriate program for execution of the operator. Figure 3-4 shows the flow within subsystem 4.

For each operator, the operand is classified by the appropriate program and, where data is specified, the data is located. Execution checks are performed and the operator is executed. Storage allocation for data may be required in the execution of operators of all classes except 4, 5 (subclass 4), 7, and 8. Storage allocation for user programs can only occur when a class 2 operation is being executed.

When an error is detected, the subsystem terminates execution of the interpreter instruction, follows the established program linkage back to the conversational mode program stored in the program construction area {1.1}, releases any data not associated with the conversational mode program, and prints the required error message. Error messages requested by other subsystems or programs are also provided by subsystem 4.

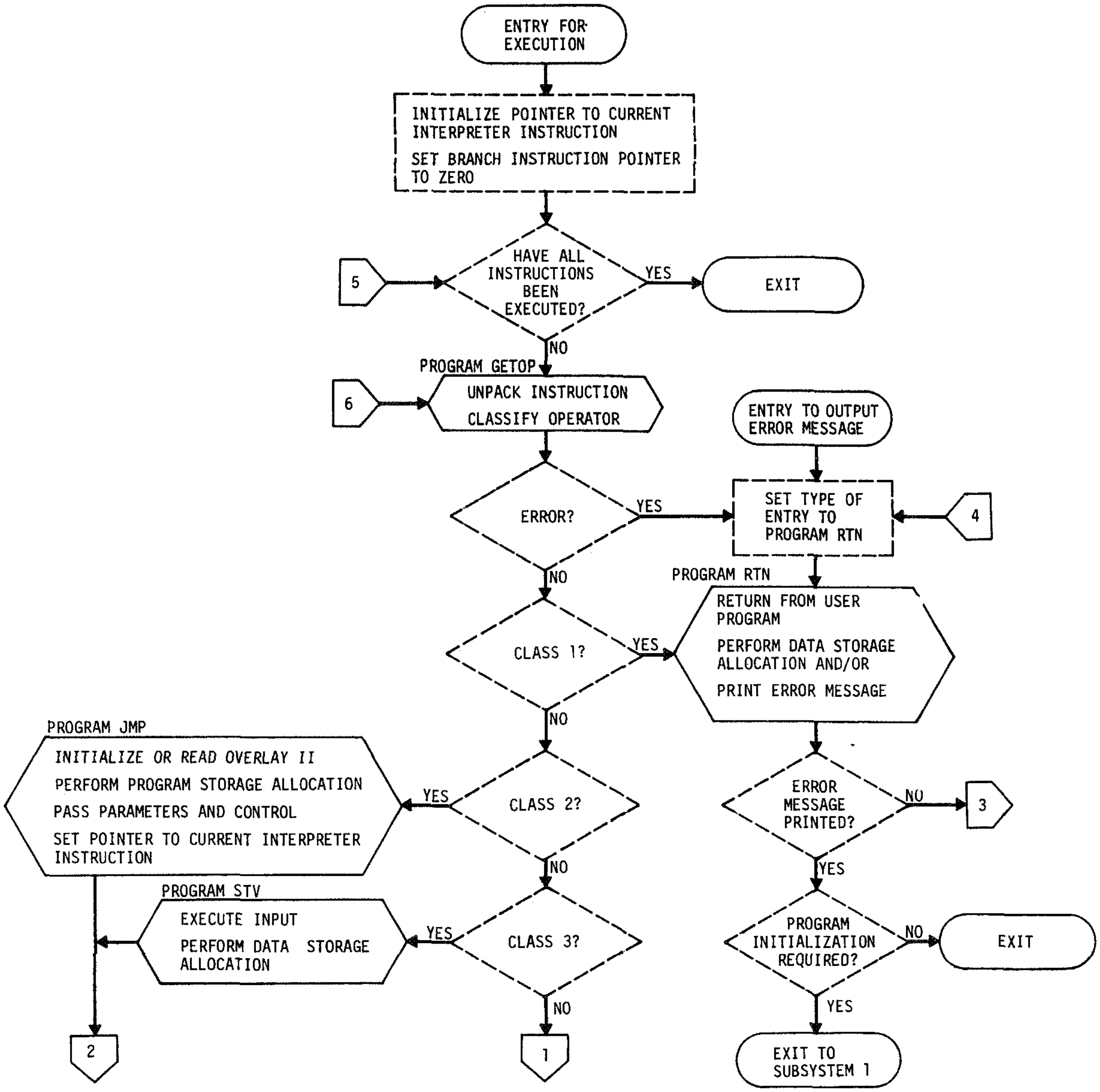

ENTRY FOR EXECUTION
INITIALIZE POINTER TO CURRENT INTERPRETER INSTRUCTION
SET BRANCH INSTRUCTION POINTER TO ZERO
5
HAVE ALL INSTRUCTIONS BEEN EXECUTED?
YES
EXIT
NO
PROGRAM GETOP
6
UNPACK INSTRUCTION
CLASSIFY OPERATOR
ENTRY TO OUTPUT ERROR MESSAGE
ERROR?
YES
SET TYPE OF ENTRY TO PROGRAM RTN
4
NO
PROGRAM RTN
RETURN FROM USER PROGRAM
PERFORM DATA STORAGE ALLOCATION AND/OR
PRINT ERROR MESSAGE
CLASS 1?
YES
NO
PROGRAM JMP
INITIALIZE OR READ OVERLAY II
PERFORM PROGRAM STORAGE ALLOCATION
PASS PARAMETERS AND CONTROL
SET POINTER TO CURRENT INTERPRETER INSTRUCTION
YES
CLASS 2?
ERROR MESSAGE PRINTED?
NO
3
NO
YES
PROGRAM STV
EXECUTE INPUT
PERFORM DATA STORAGE ALLOCATION
YES
CLASS 3?
PROGRAM INITIALIZATION REQUIRED?
NO
EXIT
NO
YES
2
1
EXIT TO SUBSYSTEM 1

FIGURE 3-4. SUBSYSTEM 4

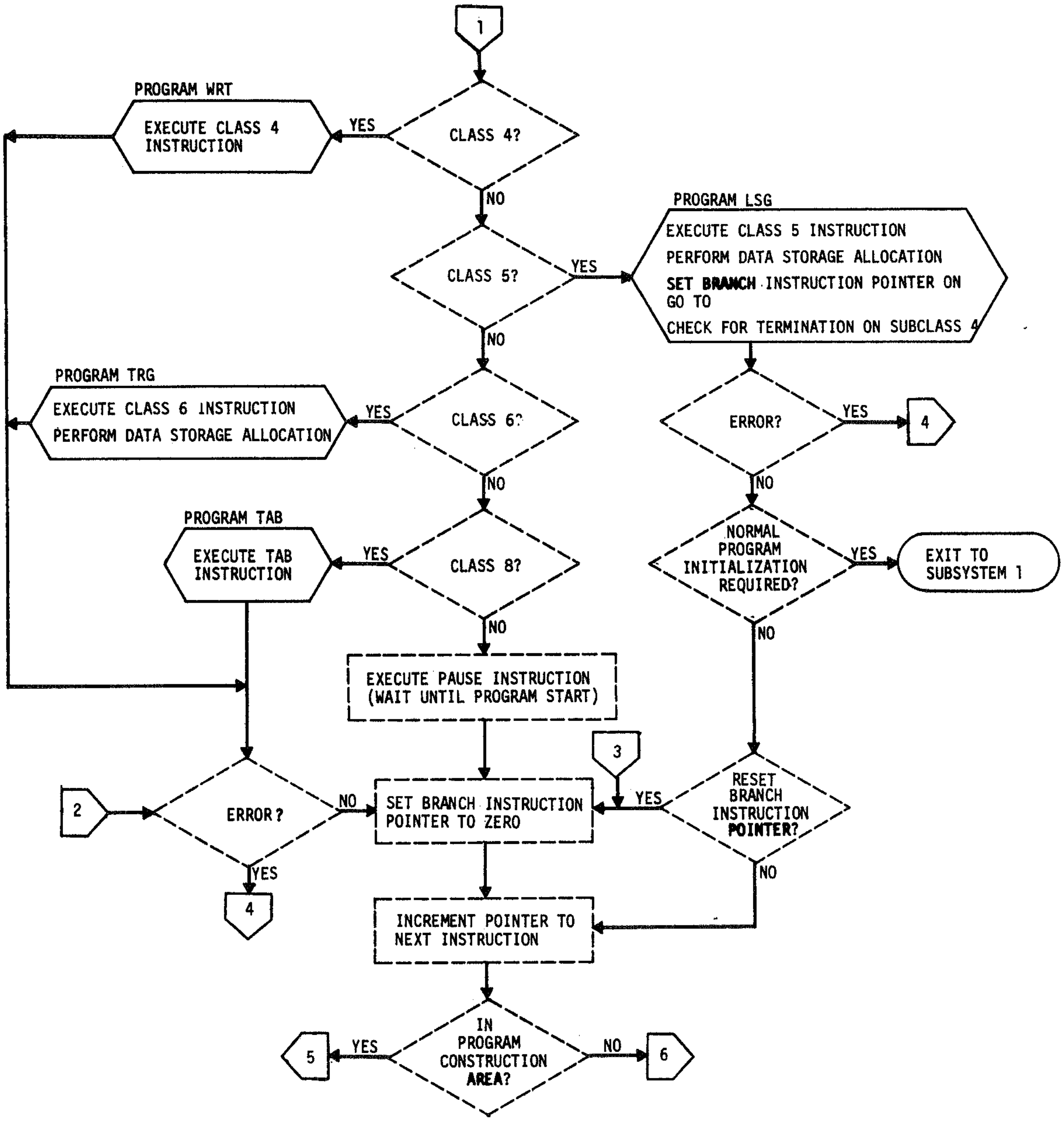

1
PROGRAM WRT
EXECUTE CLASS 4 INSTRUCTION
YES
CLASS 4?
NO
PROGRAM LSG
EXECUTE CLASS 5 INSTRUCTION
PERFORM DATA STORAGE ALLOCATION
SET BRANCH INSTRUCTION POINTER ON GO TO
CHECK FOR TERMINATION ON SUBCLASS 4
CLASS 5?
YES
NO
PROGRAM TRG
EXECUTE CLASS 6 INSTRUCTION
PERFORM DATA STORAGE ALLOCATION
YES
CLASS 6?
NO
ERROR?
YES
4
NO
PROGRAM TAB
EXECUTE TAB INSTRUCTION
YES
CLASS 8?
NO
NORMAL PROGRAM INITIALIZATION REQUIRED?
YES
EXIT TO SUBSYSTEM 1
NO
EXECUTE PAUSE INSTRUCTION (WAIT UNTIL PROGRAM START)
3
2
ERROR?
NO
SET BRANCH INSTRUCTION POINTER TO ZERO
YES
RESET BRANCH INSTRUCTION POINTER?
YES
4
NO
INCREMENT POINTER TO NEXT INSTRUCTION
5
YES
IN PROGRAM CONSTRUCTION AREA?
NO
6

FIGURE 3-4 - Concluded

4. PROGRAM DESCRIPTIONS

Detailed descriptions of each program in the system are provided in this section. Many of the programs perform extensive error checks; however, in order to maintain clarity in the explanations of the programs' major tasks, the error checks are not detailed in each program. Appendix A contains a list of the error messages and a table showing which programs can request the output of each message. Program listings are provided in Appendix B.

4.1 PROGRAM CTL

The Assembler language program CTL is the control program of the entire system. It is called by the main program AMTRN which exists only to take advantage of the disk file and COMMON definition capabilities of FORTRAN. Program CTL performs all of the functions shown in the functional flowchart, Figure 2-3, and the functions which appear in dotted lines in the flowcharts of the four subsystems, Figures 3-1, 3-2, 3-3, and 3-4. By checking indicators and parameters set by the various programs, program CTL handles all routing of the system flow between subsystems and programs within subsystems. Program CTL directly calls the following programs, which cannot directly call one another:

Subsystem	Program
	ITZ
	LST
1	RST NAM EDT DLT
2	RDLL SCA SCB

Subsystem	Program
3	STK CDR
4	GETOP RTN JMP STV WRT LSG TRG TAB.

The organization of the control program and the programs of the system is imposed by the characteristics of the IBM 1130 Disk Monitor System and by core limitations.

4.2 PROGRAM ITZ

The program ITZ, flowcharted in Figure 4-1, initializes the system and is executed only once. The program checks the file status word {21.1} on the user program file {21.} to determine if the file is packed or unpacked, and repacks the file if required. After packing the file to remove any unused records between programs, the file status word is set to zero, and the updated pointer to the first available record {21.2} and the updated user program table {21.4} are written on the file. Two control arrays used by program SCB in subsystem 2 are initialized in COMMON. The arrays take up unused space in the variable table {1.2-1.5}, namely, the first and second columns of rows 32 through 43. The system constants {1.3.2} are also initialized. The EBCDIC table {1.2-1.4} is read into COMMON from the system control file {22.4} and initialized on the system working file {20.6}. The card status {1.1.4}, overlay status {3.8}, statement entry mode {16.}, and the type of entry to program RST {3.4} are set to one; the variable {15.} and previous variable {1.1.3} counts are set to zero.

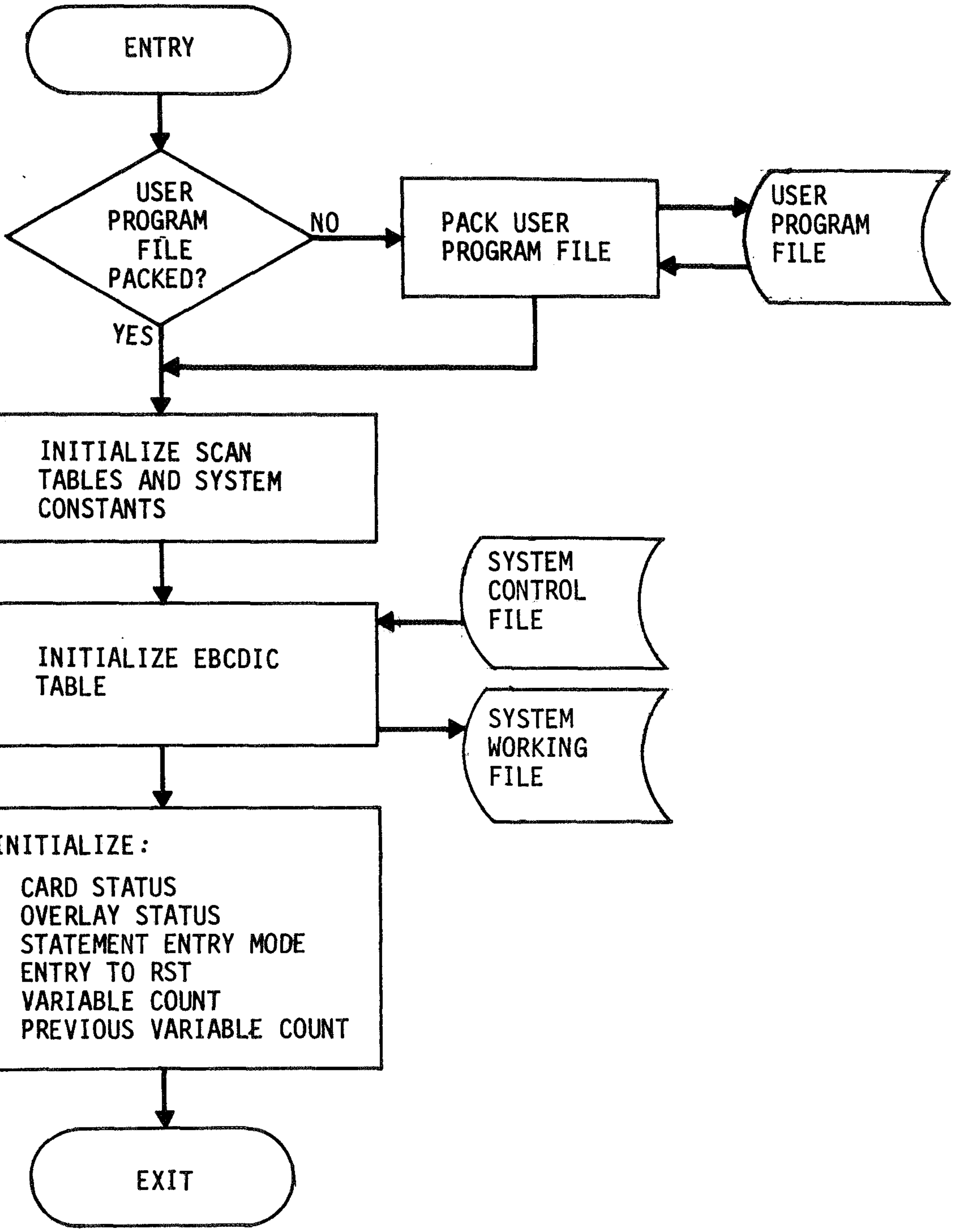


FIGURE 4-1. PROGRAM ITZ

4.3 SUBSYSTEM 1

4.3.1 Program RST

The major tasks of this program are the initialization of the system to accept a new program and the restoration of a previous mode program for continued entry of statements. The functioning of RST is controlled mainly by the type of entry {3.4} which indicates the following:

1 - Entry from program ITZ, program EDT, subsystem 2, or subsystem 4, indicating program initialization requirement

2 - Entry from program DLT, indicating a program has been deleted from the user program file {21.}

3.- Entry from program NAM, indicating a program has been stored on the user program file

4 - Entry from subsystem 2, indicating a change of statement entry mode requirement

5 - Entry from program DLT, indicating storage in the user data area {1.3.1} has been freed.

The types of entry to RST are discussed below in the order in which they appear in the flowchart, Figure 4-2.

4.3.1.1 Entry Type 3 - Program RST updates the user program table {21.4} after the program NAM has stored a user program on the user program file {21.}. The program name and file storage information are provided for RST in six words, beginning in KODE(513). The six words contain the information to be entered directly into the program control table:

Subscript in KODE	Contents
513, 514, 515	Program name (2 character per word)
516	Record number at which internal form of program begins on the file

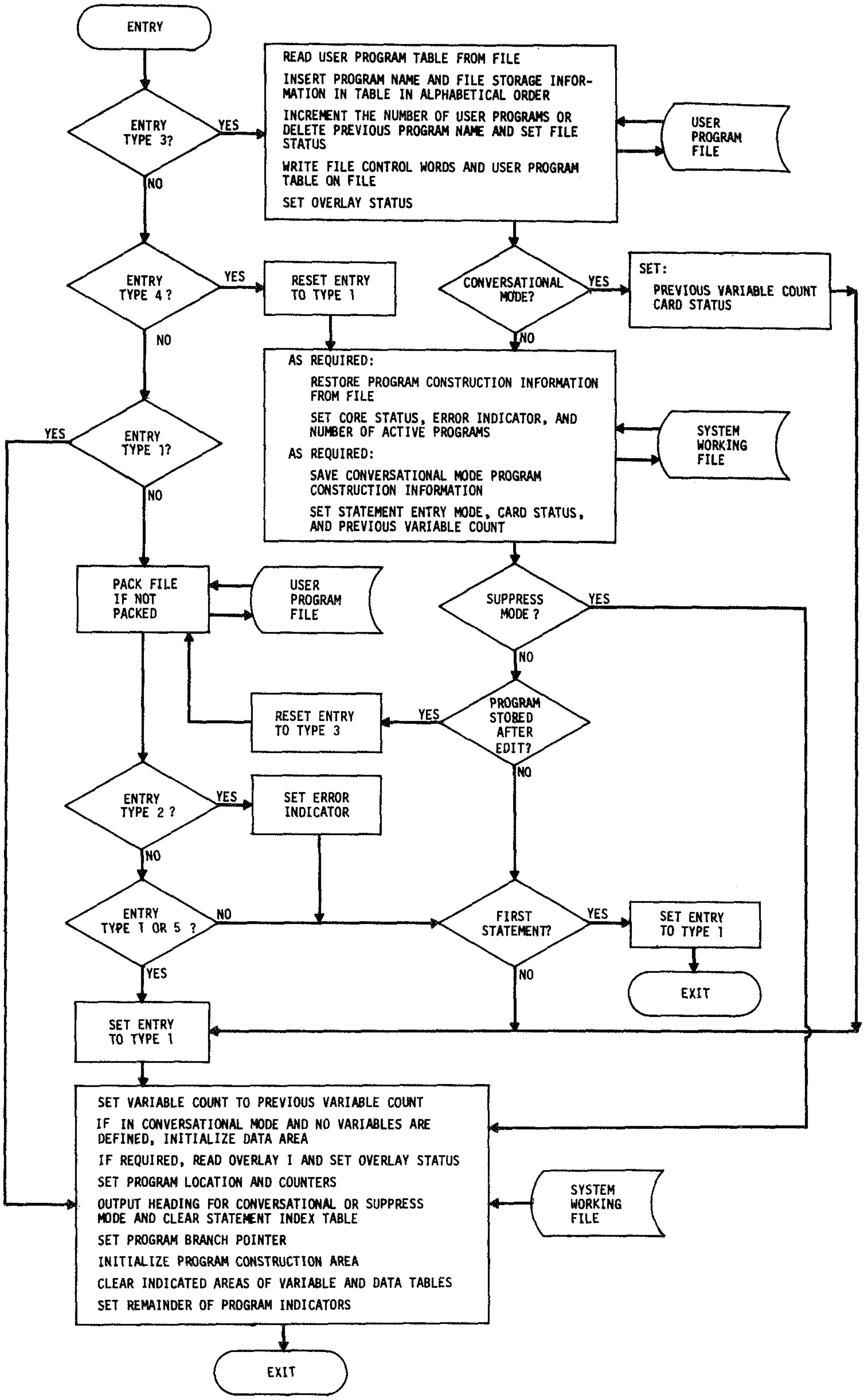

ENTRY
ENTRY TYPE 3?
YES
NO
READ USER PROGRAM TABLE FROM FILE
INSERT PROGRAM NAME AND FILE STORAGE INFORMATION IN TABLE IN ALPHABETICAL ORDER
INCREMENT THE NUMBER OF USER PROGRAMS OR DELETE PREVIOUS PROGRAM NAME AND SET FILE STATUS
WRITE FILE CONTROL WORDS AND USER PROGRAM TABLE ON FILE
SET OVERLAY STATUS
USER PROGRAM FILE
ENTRY TYPE 4 ?
YES
NO
RESET ENTRY TO TYPE 1
CONVERSATIONAL MODE?
YES
NO
SET:
PREVIOUS VARIABLE COUNT
CARD STATUS
AS REQUIRED:
RESTORE PROGRAM CONSTRUCTION INFORMATION FROM FILE
SET CORE STATUS, ERROR INDICATOR, AND NUMBER OF ACTIVE PROGRAMS
AS REQUIRED:
SAVE CONVERSATIONAL MODE PROGRAM CONSTRUCTION INFORMATION
SET STATEMENT ENTRY MODE, CARD STATUS, AND PREVIOUS VARIABLE COUNT
SYSTEM WORKING FILE
ENTRY TYPE 1?
YES
NO
PACK FILE IF NOT PACKED
USER PROGRAM FILE
SUPPRESS MODE ?
YES
NO
PROGRAM STORED AFTER EDIT?
YES
NO
RESET ENTRY TO TYPE 3
ENTRY TYPE 2 ?
YES
NO
SET ERROR INDICATOR
ENTRY TYPE 1 OR 5 ?
NO
YES
FIRST STATEMENT?
YES
NO
SET ENTRY TO TYPE 1
EXIT
SET ENTRY TO TYPE 1
SET VARIABLE COUNT TO PREVIOUS VARIABLE COUNT
IF IN CONVERSATIONAL MODE AND NO VARIABLES ARE DEFINED, INITIALIZE DATA AREA
IF REQUIRED, READ OVERLAY I AND SET OVERLAY STATUS
SET PROGRAM LOCATION AND COUNTERS
OUTPUT HEADING FOR CONVERSATIONAL OR SUPPRESS MODE AND CLEAR STATEMENT INDEX TABLE
SET PROGRAM BRANCH POINTER
INITIALIZE PROGRAM CONSTRUCTION AREA
CLEAR INDICATED AREAS OF VARIABLE AND DATA TABLES
SET REMAINDER OF PROGRAM INDICATORS
SYSTEM WORKING FILE
EXIT

FIGURE 4-2. PROGRAM RST

Subscript in KODE	Contents
517	Record number at which the source statement count is stored
518	Number of records occupied by statement count, statement index table, and source statements.

The program table is read from the user program file and the program name and storage information are entered into the table, keeping the table in alphabetical order by program name. If the program was entered into the table as the result of an EDIT operation and if the program name is different from the previous name, the previous entry for the program is deleted from the table. The requirement for this action is determined by the indicator {3. 1-1} which was set on the initial EDIT statement to the row in the program table {21. 4} containing the information for the program being edited. If the system is in the EDIT mode, the file status word {21. 1} is set to one. If the program was named in the conversational or SUPPRESS mode, the number of console programs {14. } and {21. 3} is incremented. The updated program table {21. 4} is written on the user program file. The overlay status {3. 8} is set to two.

If the system is in the conversational mode, the previous variable count {1. 1. 3} is set to zero, the card status {1. 1. 4} and the type of entry to RST {3. 4} are set to one, and the program functions proceed as for an entry type 1 (see Section 4. 3. 1. 4).

If the system is not in the conversational mode, the program functions for an entry type 4 are executed, omitting the setting of the entry type to one.

4. 3. 1. 2 Entry Type 4 - The first action taken on this type of entry is to reset the entry type to one. Program RST then determines and processes any change of statement entry mode required by the following user commands:

- NAME (only if the program was entered in the SUPPRESS or EDIT mode and has already been stored on the user program file {21.})

- SUPPRESS

- END (when entered in the SUPPRESS or EDIT mode)

- CARD.

A change of entry mode may require the storage or retrieval of the following program construction information:

- Program construction area {1. 1}

- Overlay I {1. 2-1}, except the statement conversion area {1. 2-1. 1}

- All remaining variables named in COMMON, except the data table {2.} and the number of user programs {14.}.

Table 4-1 shows the program information read or write requirements and the area involved on the system working file {20.} as a function of the user command and the statement entry mode {16.}.

If the system enters the SUPPRESS mode in response to a SUPPRESS or CARD command, the program RST ensures that the statement entry mode is set to two, that the previous variable count {1. 1. 3} is set to zero, and that the card status {1. 1. 4} is set to one for a SUPPRESS command and to three for a CARD command. The program then performs the actions of entry type 1 (see Section 4. 3. 1. 4).

For all other commands, the overlay status {3. 8} is set to one, the error indicator {9.} is set positive, and the active program count {19.} is set to zero. If the command was NAME and occurred in the EDIT mode, the type of entry to RST {3. 4} is set to three and the functions described for entry types 2 and 5 are performed. Otherwise, the type of entry to program RST is set to one and one of two alternate actions are taken depending on the value of the statement count {18.}: if the count

TABLE 4-1. PROGRAM CONSTRUCTION INFORMATION READ OR WRITE REQUIREMENTS

Command	Conversational Mode	Suppress Mode	Edit Mode
NAME		Read {20.4}	Read {20.5}
SUPPRESS	Write {20.4}		Read {20.5}*
END		Read {20.4}	
CARD	Write {20.4}		Read {20.5}*

*The restored mode is reprocessed as if it were the mode in which the command was entered.

is one, exit from RST; if the count is not one, perform functions of entry type 1 (see Section 4.3.1.4).

4.3.1.3 Entry Types 2 and 5 - These entries indicate that the user program file {21.} may require repacking. The file status word {21.1} is checked. If the file is not packed, the program RST packs the file, removing any unused records between programs, resetting the file status word to zero, and writing the updated pointer to the first available record {21.2} and the updated user program table {21.4} on the file. The type of entry to RST {3.4} is set to one. If the original entry was type 5, the functions of entry type 1 are performed. If the original entry was type 2, the error indicator {9.} is set positive. For entry type 2 and an entry to this section after a completion of an editing function (see Section 4.3.1.2), the statement count {18.} is checked. If the value is one, an exit from RST is made; otherwise, the functions for an entry type 1 are executed.

4.3.1.4 Entry Type 1 - This entry is for normal program and data initialization. The variable count {15.} is set equal to the previous variable count {1.1.3}. If the variable count is zero and the statement entry mode {16.} is one, data initialization is performed. This consists of setting the data table entry count {10.} to zero, the data storage count {12.} to 604, and the initial values listed under the user data area {1.3.1} description.

If the overlay status {3.8} is two, Overlay I {1.2-1} is read from the system working file {20.6} and the overlay status is set to one. RST then sets the current program pointer {8.} and the source statement record count {3.6} to one and the active program count {19.}, the user constant count {11.}, the statement count {18.}, the error indicator {9.}, the previous program count {1.1.2}, and the program count {1.1.1} to zero.

The pointers to the last interpreter instruction generated for the previous statement {13.} and the current statement {6.} are initialized to seven. The first three words of KODE are set to 252, 302, and 410, respectively. The rows of the variable table {1.2-1.5} which do not contain names of variables are cleared by setting them to zero. In the rows used for temporary variables, column five is set to zero and column four is set to minus one. If the system is in the conversational mode, unused rows of the data table {2.} are set to zero. Entries not required in the variable linkage area {1.1.6} are set to zero.

If the system is in the conversational or SUPPRESS mode, the heading

ENTER PROGRAM

or

ENTER PROGRAM - SUPPRESSED

is printed on the console typewriter. Also, the statement index table {1.2-1.3} is set to all zeros.

4.3.2 Program NAM

In response to the NAME statement, the program NAM stores a user program which has been entered in the conversational, SUPPRESS, or EDIT mode on the user program file {21.}. The functions performed by the program NAM can be grouped into eight tasks. The tasks are described below and the program is flowcharted in Figure 4-3.

4.3.2.1 Task 1 - Column 2 of the current row of the statement index table {1.2-1,3} is placed in form 2. The current row is specified by the statement count {18.}. The value added to column 2 is determined by the pointer to the last interpreter instruction generated for the previous statement {13.} and by the program branch pointer {1.2-1.2}.

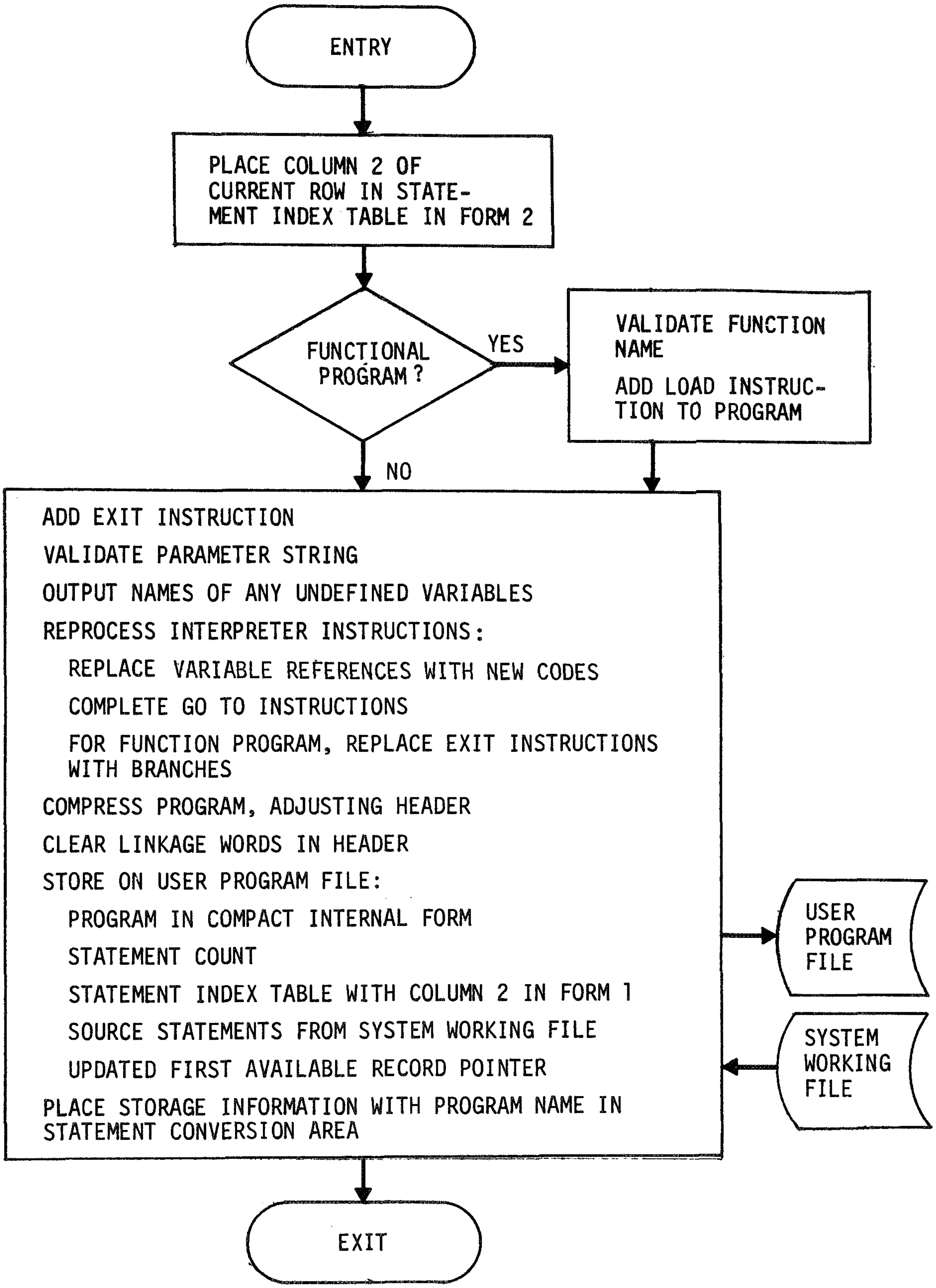

ENTRY
PLACE COLUMN 2 OF CURRENT ROW IN STATEMENT INDEX TABLE IN FORM 2
FUNCTIONAL PROGRAM?
YES
VALIDATE FUNCTION NAME
ADD LOAD INSTRUCTION TO PROGRAM
NO
ADD EXIT INSTRUCTION
VALIDATE PARAMETER STRING
OUTPUT NAMES OF ANY UNDEFINED VARIABLES
REPROCESS INTERPRETER INSTRUCTIONS:
REPLACE VARIABLE REFERENCES WITH NEW CODES
COMPLETE GO TO INSTRUCTIONS
FOR FUNCTION PROGRAM, REPLACE EXIT INSTRUCTIONS WITH BRANCHES
COMPRESS PROGRAM, ADJUSTING HEADER
CLEAR LINKAGE WORDS IN HEADER
STORE ON USER PROGRAM FILE:
PROGRAM IN COMPACT INTERNAL FORM
STATEMENT COUNT
STATEMENT INDEX TABLE WITH COLUMN 2 IN FORM 1
SOURCE STATEMENTS FROM SYSTEM WORKING FILE
UPDATED FIRST AVAILABLE RECORD POINTER
PLACE STORAGE INFORMATION WITH PROGRAM NAME IN STATEMENT CONVERSION AREA
USER PROGRAM FILE
SYSTEM WORKING FILE
EXIT

FIGURE 4-3. PROGRAM NAM

4.3.2.2 Task 2 - The NAME statement is passed to NAM from subsystem 2 in state 1 of the statement conversion area {1.2-1.1} with the following information in the indicated locations:

Subscript in KODE	Contents
511	Internal code for NAME label
512	Variable internal code or zero
513, 514, 515	Program name (2 characters per word)
516	Unused
517	Beginning of argument list.

If KODE (512) is not zero, the user has defined a function program by using a program name which is also a variable name. This word contains the internal code for the variable. If the program is a function, it is verified that the variable is defined in the program. A load accumulator instruction (see Table 2-4) with the variable as the operand is added to the program interpreter instructions area {1.1.5}.

4.3.2.3 Task 3 - The instruction to exit from the program is added to the interpreter instruction area {1.1.5}.

4.3.2.4 Task 4 - If the user declared an argument list in the NAME statement, the parameter string begins in KODE (517) and is a mixture of internal codes for variables (see Section 4.4.2.2) and delimiters in AMTRAN character codes (see Table 4-2). The parameter string must be enclosed in parentheses; the variables must be separated by commas; and the string must end with period - eof. The parameter string is scanned and the variables are assigned new internal codes in the order in which they occur; the first variable is assigned the number 401.

4.3.2.5 Task 5 - The entries in the variable table {1.2-1.5} which are not referenced in the parameter string are checked to determine if they appear to be undefined (column 4 is zero and column 5 is negative). The names of all undefined variables are printed on the typewriter. All entries not referenced in the parameter string are assigned new internal codes beginning with the first 400 number after the parameter variables (see Section 4.3.2.4). Next, new 400 codes beginning with the next sequential number are assigned to any temporaries used by the program (column 4 is one).

4.3.2.6 Task 6 - All of the interpreter instructions {1.1.5} generated for the program are reprocessed. The last instruction pointer {6.} provides the location of the last instruction. In the reprocessing, all variable codes are replaced with the newly assigned codes. This makes all variable and temporary references contiguous numbers beginning with 401. All branch instructions are completed. When a forward branch is entered in the SUPPRESS mode, subsystem 2 replaces the statement number with 449 plus the row for that statement number in the statement index table {1.2-1.3} (see Section 4.4.3.1). The program NAM replaces this reference number with the correct displacement to accomplish the branch. If the program is a function, all exit instructions except the last are replaced with branches to the load accumulator instruction added in task 2 (see Section 4.3.2.2).

4.3.2.7 Task 7 - The program is packed by removing unused positions in the interpreter instruction area {1.1.5}, variable linkage area {1.1.6}, user constants area {1.1.7}, and user program call table {1.1.8}. Since the user constant area must begin in an odd subscript to access the constants in floating point, an unused word may occur in the variable linkage area. As the program is packed, the first three words of the program header are adjusted (see Figure 2-1). The fourth word of the header is set to the number of parameters detected by task 4. Words 5, 6, and 7 of the header are set to zero.

4.3.2.8 Task 8 - The compacted program is written on the user program file {21.}, beginning at the record specified as the next available {21.2}. The statement count {18.} is written on the file behind the program. That portion of the statement index table {1.2-1.3} containing entries is written next with column 2 converted back to form 1. The source statements for the program are transferred to the program file from the system working file (either {20.1}, {20.2}, or {20.3}, depending on the statement entry mode {16.}). The source statements remain packed two characters per word. The first available record {21.2} is updated on the file. The file storage information required for the program control table {21.4} is entered in KODE in the three words following the program name (see Section 4.3.2.2). The type of entry to the program RST {3.4} is set to three.

4.3.3 Program EDT

Program EDT, shown in Figure 4-4, initializes the statement index table {1.2-1.3} and several indicators and counters in preparation for editing a user program. Subsystem 2 recognizes the EDIT command, locates the program to be edited on the program file {21.}, and provides the following information to program EDT:

- EDIT statement in state 1 of statement conversion area {1.2-1.1}
- Working register {4-2} - set to displacement relative to KODE (510) at which the scan for statement numbers is to begin
- FORTRAN record control {7.} - set to the record on the user program file {21.} which contains the statement count for the program.

Program EDT sets the active program count {19.} to zero and reads the statement count and the statement index table from the program file. The input string is scanned, converting each statement number to the same format as the numbers in column 1 of the table. If the statement

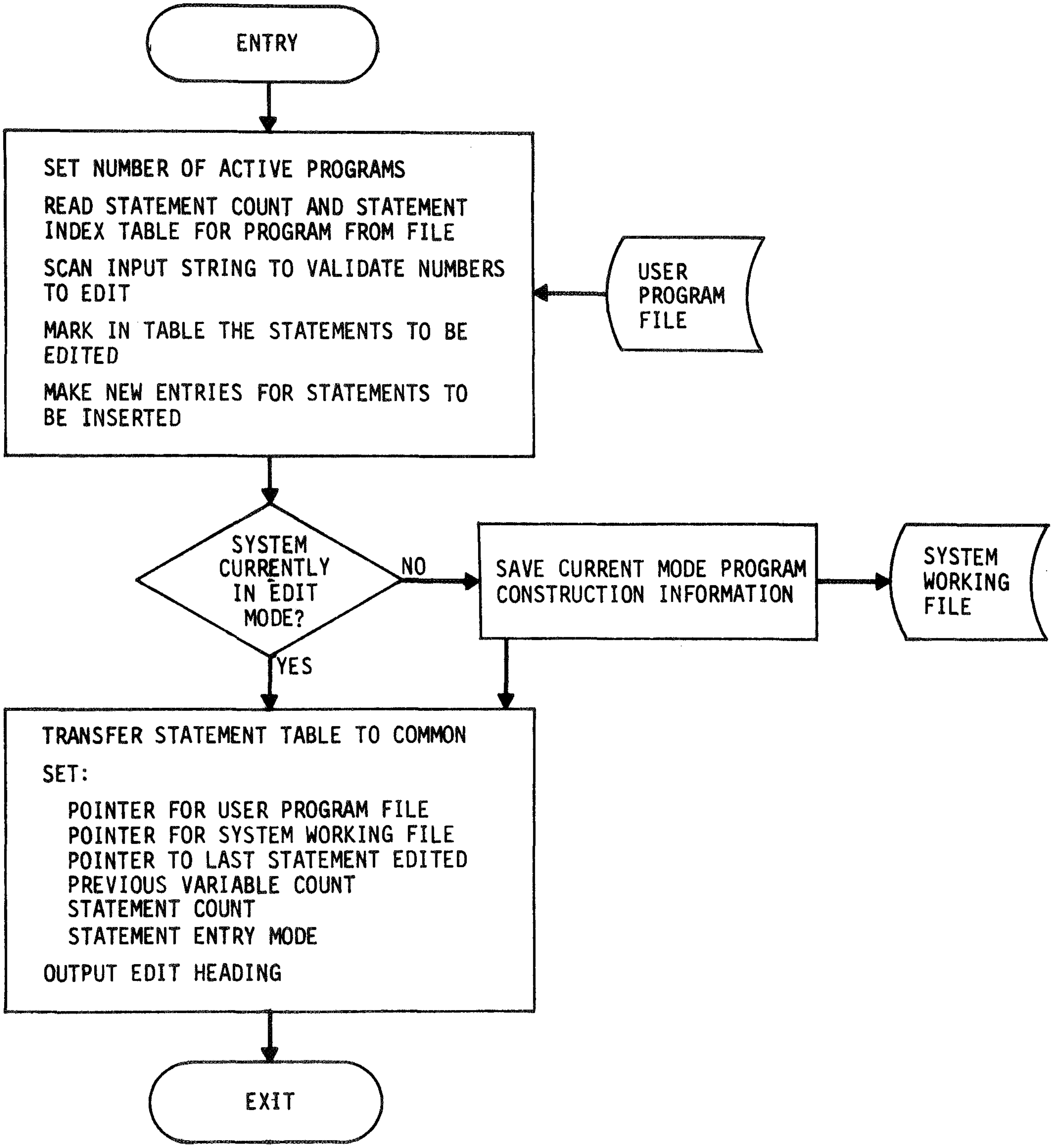

ENTRY
SET NUMBER OF ACTIVE PROGRAMS
READ STATEMENT COUNT AND STATEMENT INDEX TABLE FOR PROGRAM FROM FILE
SCAN INPUT STRING TO VALIDATE NUMBERS TO EDIT
MARK IN TABLE THE STATEMENTS TO BE EDITED
MAKE NEW ENTRIES FOR STATEMENTS TO BE INSERTED
USER PROGRAM FILE
SYSTEM CURRENTLY IN EDIT MODE?
NO
SAVE CURRENT MODE PROGRAM CONSTRUCTION INFORMATION
SYSTEM WORKING FILE
YES
TRANSFER STATEMENT TABLE TO COMMON
SET:
POINTER FOR USER PROGRAM FILE
POINTER FOR SYSTEM WORKING FILE
POINTER TO LAST STATEMENT EDITED
PREVIOUS VARIABLE COUNT
STATEMENT COUNT
STATEMENT ENTRY MODE
OUTPUT EDIT HEADING
EXIT

FIGURE 4-4. PROGRAM EDT

number appears in the table, the sign of the corresponding entry in column 2 (which is the source statement length) is reversed to indicate the statement is to be edited. If the statement number does not appear in the table, but is within the number range, a new entry is inserted into the table, keeping the statement numbers in sequential order. The corresponding entry in column 2 is set to zero.

After completion of the scan, the statement entry mode {16.} is checked. If the system is currently in the conversational or the SUPPRESS mode, the following program construction information for the current mode is stored on the system file {20.5-1}:

- Program construction area {1.1}
- Overlay I {1.2-1}, except the statement conversion area {1.2-1.1}
- All other variables named in COMMON, except the data table {2.} and the number of user programs {14.}.

The modified statement index table is transferred to COMMON {1.2-1.3} and the following are initialized for editing:

- Source statement record pointer {3.2-1} - set to record at which first source statement begins on the user program file
- Source statement record count {3.6} - set to one
- Pointer to last statement edited {3.3} - set to zero
- Previous variable count {1.1.3} - set to zero
- Statement count {18.} - set to zero
- Statement entry mode {16.} - set to three.

As a final task, the label EDIT is output on the typewriter.

4.3.4 Program DLT

Program DLT, shown in Figure 4-5, makes deletions on the user program file {21.} and in the user data area {1.3.1} in response

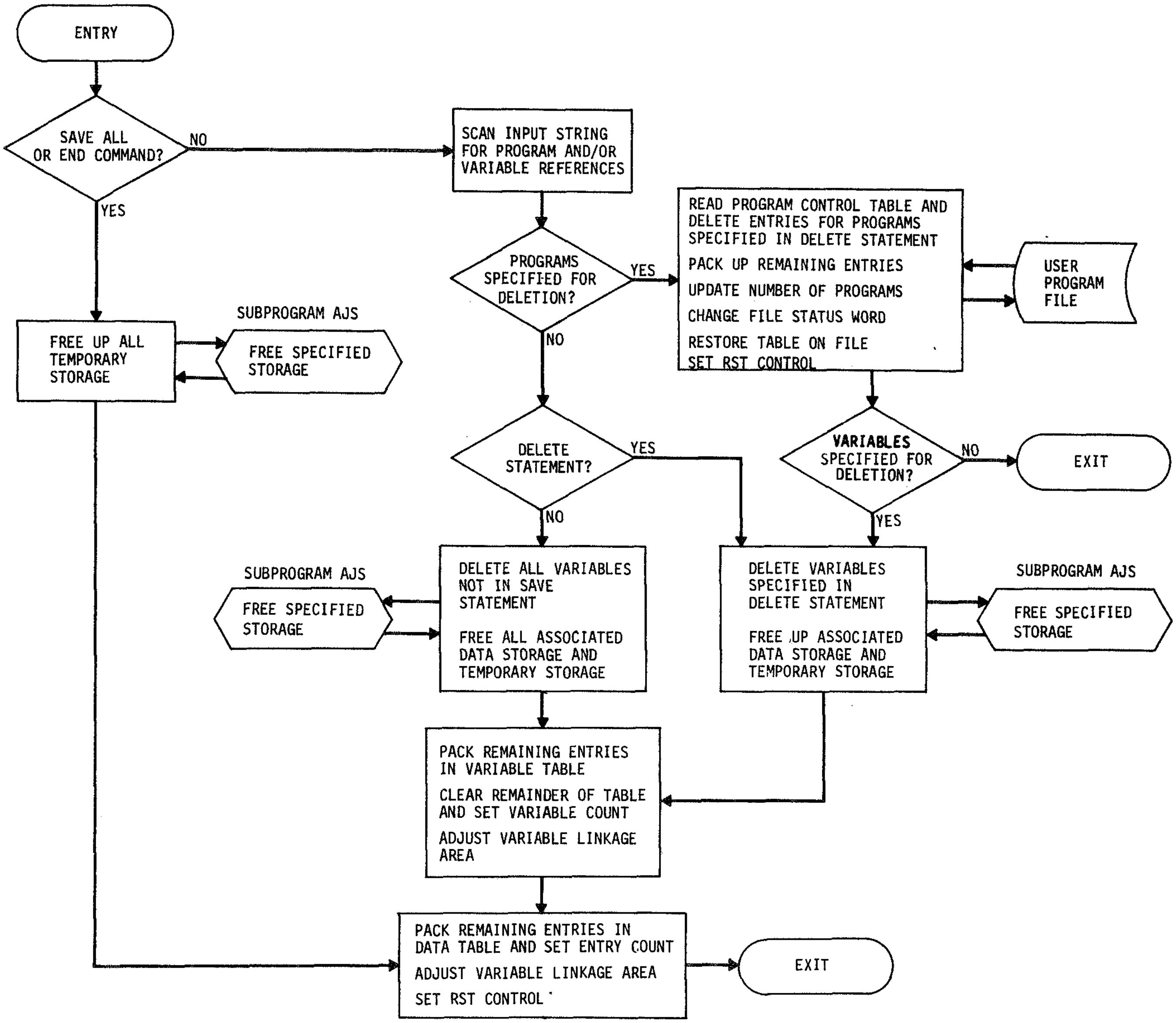

ENTRY
SAVE ALL OR END COMMAND?
NO
YES
SCAN INPUT STRING FOR PROGRAM AND/OR VARIABLE REFERENCES
PROGRAMS SPECIFIED FOR DELETION?
YES
NO
READ PROGRAM CONTROL TABLE AND DELETE ENTRIES FOR PROGRAMS SPECIFIED IN DELETE STATEMENT
PACK UP REMAINING ENTRIES
UPDATE NUMBER OF PROGRAMS
CHANGE FILE STATUS WORD
RESTORE TABLE ON FILE
SET RST CONTROL
USER PROGRAM FILE
SUBPROGRAM AJS
FREE UP ALL TEMPORARY STORAGE
FREE SPECIFIED STORAGE
DELETE STATEMENT?
YES
NO
VARIABLES SPECIFIED FOR DELETION?
NO
EXIT
YES
SUBPROGRAM AJS
FREE SPECIFIED STORAGE
DELETE ALL VARIABLES NOT IN SAVE STATEMENT
FREE ALL ASSOCIATED DATA STORAGE AND TEMPORARY STORAGE
DELETE VARIABLES SPECIFIED IN DELETE STATEMENT
FREE UP ASSOCIATED DATA STORAGE AND TEMPORARY STORAGE
SUBPROGRAM AJS
FREE SPECIFIED STORAGE
PACK REMAINING ENTRIES IN VARIABLE TABLE
CLEAR REMAINDER OF TABLE AND SET VARIABLE COUNT
ADJUST VARIABLE LINKAGE AREA
PACK REMAINING ENTRIES IN DATA TABLE AND SET ENTRY COUNT
ADJUST VARIABLE LINKAGE AREA
SET RST CONTROL
EXIT

FIGURE 4-5. PROGRAM DLT

to the commands SAVE, DELETE, SAVE ALL, and END. The commands are recognized by subsystem 2 which sets the type of entry to program DLT {3.7-1} to one of the following values:

1 - SAVE or DELETE command
2 - SAVE ALL or END command.

On a SAVE or DELETE command, the program DLT will scan the statement string which appears in state 2 of the statement conversion area {1.2-1.1} to determine the programs and/or variables to be deleted or the variables to be saved. The statement string is in internal codes, with variable and/or program references separated by commas which have internal code 268. The string is terminated by the number 99. The actual program names appear in the program call table {1.1.8}. A program reference code appearing in the input string is 100 plus the column in the table which contains the program name. Similarly, a variable reference code is 400 plus the number of the row in the variable table {1.2-1.5} in which the variable name is stored.

The user program file {21.} is affected only by a DELETE statement containing program references. The specified programs are deleted from the user program table {21.4} on the file. The file status word {21.1} is set to one; the number of user programs {14.} and {21.3} is updated; and the type of entry to RST {3.4} is set to four.

The SAVE ALL and END commands and the DELETE and SAVE statements containing variable references will cause program DLT to free up all data storage currently being used for temporary results and for the system accumulator. Temporary storage is in use if any nonzero entries exist in any of the 30th through 50th positions of the variable linkage area {1.1.6} and if a nonzero entry points to a row in the data table {2.} which also has nonzero entries. Storage for the system accumulator is allocated if the 90th row of the data table is not zero. When storage is freed by program DLT, the corresponding row in the data table and the variable link pointing to that row are set to zero.

In addition to temporary storage and the system accumulator, those variables specified in a DELETE statement are deleted from the variable table {1.2-1.5} and the associated data storage is made available for further use. Program DLT deletes all variables in the variable table not referenced in a SAVE statement and makes the related storage available.

When entries in the variable table are deleted, the table is altered so that the remaining entries are in contiguous rows beginning in the first row. Alterations are also made to the variable linkage area {1.1.6} to maintain the one-to-one correspondence between the variable table and linkage area. The previous variable count {1.1.3} is set to the number of variable names remaining in the table.

When all data storage to be freed has been processed, the entries remaining in the data table {2.} are moved to the top of the table and the references remaining in the variable linkage area and the data table entry count {10.} are adjusted accordingly. The type of entry to program RST {3.4} is set to two.

4.4 SUBSYSTEM 2

4.4.1 Program RDLL

The main task of program RDLL is to read AMTRAN source statements. The functions involved, shown in Figure 4-6, can be divided into thirteen tasks. The tasks are executed in order, unless otherwise indicated.

4.4.1.1 Task 1 - Program RDLL ensures that Overlay I {1.2-1} is in core. If not in core, the overlay is read from the system working file {20.6}, with Overlay II {1.2-2} being first written on the system working file {20.7} if the active program count {19.} is not zero. The overlay status {3.8} is set to one.

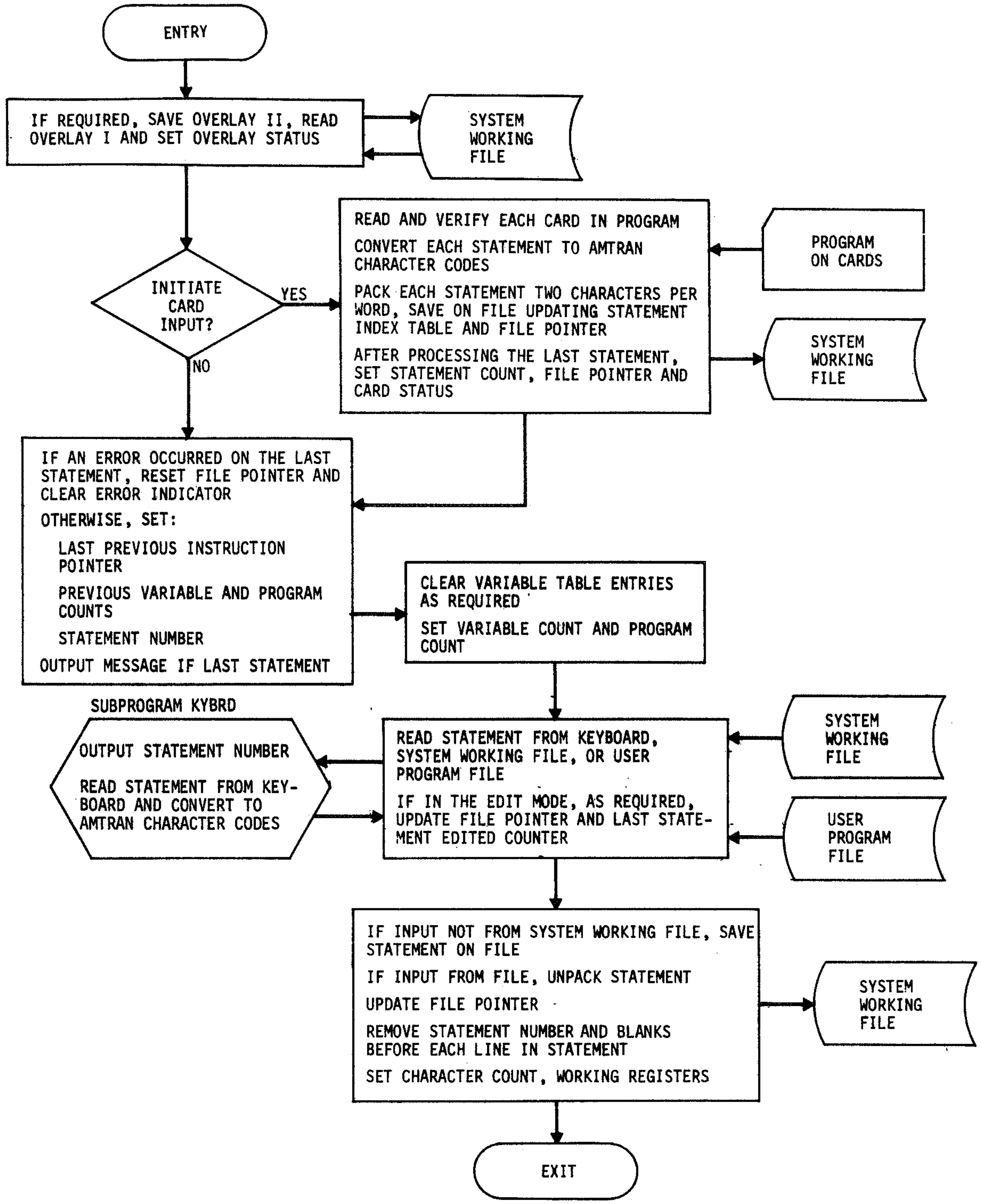

ENTRY
IF REQUIRED, SAVE OVERLAY II, READ OVERLAY I AND SET OVERLAY STATUS
SYSTEM WORKING FILE
INITIATE CARD INPUT?
YES
NO
READ AND VERIFY EACH CARD IN PROGRAM
CONVERT EACH STATEMENT TO AMTRAN CHARACTER CODES
PACK EACH STATEMENT TWO CHARACTERS PER WORD, SAVE ON FILE UPDATING STATEMENT INDEX TABLE AND FILE POINTER
AFTER PROCESSING THE LAST STATEMENT, SET STATEMENT COUNT, FILE POINTER AND CARD STATUS
PROGRAM ON CARDS
SYSTEM WORKING FILE
IF AN ERROR OCCURRED ON THE LAST STATEMENT, RESET FILE POINTER AND CLEAR ERROR INDICATOR
OTHERWISE, SET:
LAST PREVIOUS INSTRUCTION POINTER
PREVIOUS VARIABLE AND PROGRAM COUNTS
STATEMENT NUMBER
OUTPUT MESSAGE IF LAST STATEMENT
CLEAR VARIABLE TABLE ENTRIES AS REQUIRED
SET VARIABLE COUNT AND PROGRAM COUNT
SUBPROGRAM KYBRD
OUTPUT STATEMENT NUMBER
READ STATEMENT FROM KEYBOARD AND CONVERT TO AMTRAN CHARACTER CODES
READ STATEMENT FROM KEYBOARD, SYSTEM WORKING FILE, OR USER PROGRAM FILE
IF IN THE EDIT MODE, AS REQUIRED, UPDATE FILE POINTER AND LAST STATEMENT EDITED COUNTER
SYSTEM WORKING FILE
USER PROGRAM FILE
IF INPUT NOT FROM SYSTEM WORKING FILE, SAVE STATEMENT ON FILE
IF INPUT FROM FILE, UNPACK STATEMENT
UPDATE FILE POINTER
REMOVE STATEMENT NUMBER AND BLANKS BEFORE EACH LINE IN STATEMENT
SET CHARACTER COUNT, WORKING REGISTERS
SYSTEM WORKING FILE
EXIT

FIGURE 4-6. PROGRAM RDLL

4.4.1.2 Task 2 - This task is performed only if program RDLL is to initiate the input of a user program from punched cards. For each statement, the following are done:

- Increment the statement count {18.}
- Read set of cards comprising statement
- Check that statement number is valid and in sequence and enter it in column 1 of the statement index table {1.2-1.3} in the row designated by the statement count.
- Remove any trailing blanks from each card image.
- Convert the statement from EBCDIC to AMTRAN character codes checking for illegal characters. Table 4-2 contains the AMTRAN character set and the corresponding character codes.
- Perform tasks 10 and 11.

The reading of cards is terminated when the NAME statement has been processed. The source statement record count {3.6} is set to one, the statement count {18.} is set to zero, the card status {1.1.4} is set to two, and task 4 is then performed.

4.4.1.3 Task 3 - If an error occurred on the last statement (the error indicator {9.} is not zero), the source statement record count {3.6} is set back to its previous value using the column 2 entry of that row of the statement index table {1.2-1.3} specified by the statement count {18.}. The number of records on the system working file for the source statement is subtracted from the record count. The error indicator {9.} is set to zero and task 5 is performed.

4.4.1.4 Task 4 - The last previous instruction pointer {13.}, previous variable count {1.1.3}, and previous program count {1.1.2} are set to the current values, respectively, of the last instruction pointer {6.}, variable count {15.}, and program count {1.1.1}. The statement

TABLE 4-2. AMTRAN CHARACTER SET AND CODES

Character	Code	Character	Code
0	0	P	26
1	1	Q	27
2	2	R	28
3	3	S	29
4	4	T	30
5	5	U	31
6	6	V	32
7	7	W	33
8	8	X	34
9	9	Y	35
Blank	10	Z	36
A	11	*	37
B	12	/	38
C	13	+	39
D	14	-	40
E	15	&	41
F	16	=	42
G	17	(	43
H	18	)	44
I	19	.	45
J	20	,	46
K	21	;	47
L	22	$	48
M	23	'	49
N	24	End of Statement	50
O	25	Carriage Return	51

count {18.} is incremented by one. If its new value is 45 and the system is not in the edit mode or the input is not from cards, a message is output, informing the user that only one more statement may be entered.

4.4.1.5 Task 5 - If the previous variable count {1.1.3} and variable count {15.} are not equal, the rows of the variable table {1.2-1.5} indicated by the two values are set to zero.

4.4.1.6 Task 6 - The variable count {15.} and program count {1.1.1} are set, respectively, to the current values of the previous variable count {1.1.3} and the previous program count {1.1.2}.

4.4.1.7 Task 7 - This task is performed if the system is in the EDIT mode. If the statement count {18.} and the pointer to the last statement edited {3. 3} have the same value, indicating an error occurred on the last statement and the statement is to be reentered, the statement number is extracted from the row in the statement index table {1.2-1.3} specified by the statement count and control goes to task 9. Otherwise, program RDLL determines the type of entry for the current statement, depending on the status of the column 2 entry of the current row of the statement index table. The entry can indicate the following:

- Less than zero - the statement is to be reentered by the user
- Equal to zero - the statement is to be inserted into the program by the user
- Greater than zero - the statement is not to be changed.

For the first case, the source statement record pointer {3.2-1} is set to the location on the user program file {21.} of the next source statement by subtracting the table entry from the current value of the pointer. (The entry is the negative of the length of the current source statement.) The pointer to the last statement edited {3.3} is set to the value of the statement count {18.}, the statement number is extracted from the statement index table, and control goes to task 9. When the entry is zero, the same actions as described for the first case are taken with the exception of incrementing the source statement record pointer. In the last case, the source statement is read from the program file at the place specified by the source statement record pointer. The length of the statement is specified by the value of the table entry which is added to the source statement record pointer after the read. Task 11 is then performed.

4.4.1.8 Task 8 - This task is performed if the system is not in the EDIT mode. In the conversational and SUPPRESS modes, the value of the statement count {18.} is the statement number of the current statement. This value is entered into column 1 of the current row of the statement index table {1.2-1.3}.

4.4.1.9 Task 9 - The statement number is converted to a string of six characters in the form

$$d_1 d_2 \cdot d_3 d_4 b$$

where $d_1, \ldots, d_4$ are digits or blanks which replace only leading or trailing zeros in the number. The characters, in AMTRAN character codes, are placed in the first six words of an integer array which is to be used as the input array for reading the source statement from the typewriter keyboard. The program RDLL calls the subprogram KYBRD to output the statement number, read the source statement, and convert it to AMTRAN character codes. The length of the statement (stored one character per word) is returned to RDLL in the working register {5.}.

4.4.1.10 Task 10 - Program RDLL adds a blank to the end of the statement and packs the statement two characters per word, maintaining the original unpacked form. The length of the packed form is entered into column 2 (form 1) of the current row of the statement index table {1.2-1.3}.

4.4.1.11 Task 11 - The packed form of the source statement is written on the system working file in the area (either {20.1}, {20.2}, or {20.3}) specified by the statement entry mode {16.} and at the displacement in the area specified by the source statement record count {3.6}. The record count is incremented by the length of the packed statement. If the system is not in the EDIT mode, task 13 is performed next.

4.4.1.12 Task 12 - The packed source statement is unpacked into one character per word.

4.4.1.13 Task 13 - The statement number, occupying the first six characters, is removed from the source statement. For each carriage return appearing in the source statement, the next six sequential characters, which are always blanks, are removed. The statement is output in state 1 of the statement conversion area {1.2-1.1}. To initialize information for program SCA, the length of the statement is placed in KODE (932), the working register {4.} and KODE (510) are set to zero, and the working register {5.} is set to one.

4.4.2 Program SCA

Program SCA, shown in Figure 4-7, converts labels appearing in an AMTRAN source statement to internal codes, recognizing the special system labels RESET, SUPPRESS, END, NAME, EDIT, CARD, LIST, and EXPLAIN which require an exit from subsystem 2.

The source statement is input to program SCA in state 2 of the statement conversion area {1.2-1.1}. The statement appears as a string of characters in AMTRAN character codes (see Table 4-2) with one character per word. Program SCA scans along the string, processing one character at a time. For compactness, the output string is generated in the same array that contains the input statement. (This is possible

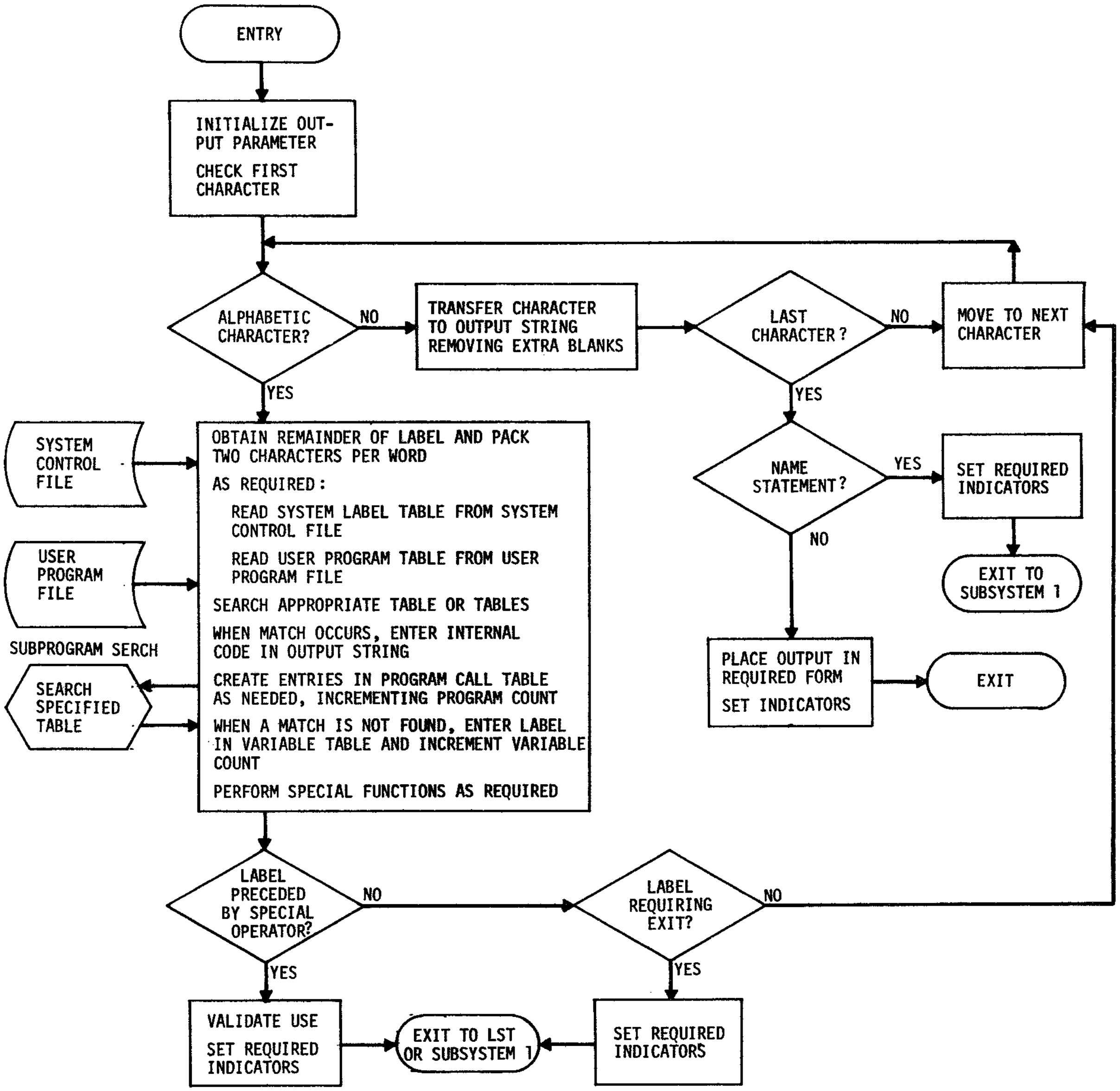

ENTRY
INITIALIZE OUTPUT PARAMETER
CHECK FIRST CHARACTER
ALPHABETIC CHARACTER?
NO
YES
TRANSFER CHARACTER TO OUTPUT STRING REMOVING EXTRA BLANKS
LAST CHARACTER?
NO
YES
MOVE TO NEXT CHARACTER
NAME STATEMENT?
YES
NO
SET REQUIRED INDICATORS
EXIT TO SUBSYSTEM 1
PLACE OUTPUT IN REQUIRED FORM
SET INDICATORS
EXIT
SYSTEM CONTROL FILE
USER PROGRAM FILE
SUBPROGRAM SERCH
SEARCH SPECIFIED TABLE
OBTAIN REMAINDER OF LABEL AND PACK TWO CHARACTERS PER WORD
AS REQUIRED:
READ SYSTEM LABEL TABLE FROM SYSTEM CONTROL FILE
READ USER PROGRAM TABLE FROM USER PROGRAM FILE
SEARCH APPROPRIATE TABLE OR TABLES
WHEN MATCH OCCURS, ENTER INTERNAL CODE IN OUTPUT STRING
CREATE ENTRIES IN PROGRAM CALL TABLE AS NEEDED, INCREMENTING PROGRAM COUNT
WHEN A MATCH IS NOT FOUND, ENTER LABEL IN VARIABLE TABLE AND INCREMENT VARIABLE COUNT
PERFORM SPECIAL FUNCTIONS AS REQUIRED
LABEL PRECEDED BY SPECIAL OPERATOR?
NO
YES
LABEL REQUIRING EXIT?
NO
YES
VALIDATE USE
SET REQUIRED INDICATORS
EXIT TO LST OR SUBSYSTEM 1
SET REQUIRED INDICATORS

FIGURE 4-7. PROGRAM SCA

since labels compress to one-word internal codes.) To reduce storage requirements, the working indicators needed by SCA are set by program RDLL (see Section 4.4.1.13).

As program SCA scans the input string, non-alphabetic characters are transferred to the output string, omitting unnecessary blanks. When an alphabetic character occurs, any additional characters are obtained, the label is classified, and an internal code representing the label is placed in the output string. The tasks involved in the label conversion will be discussed in detail. The scanning process continues until the string is totally processed or a special exit is required. When a complete statement has been scanned by program SCA, the output string is placed in state 2 of the statement conversion area and the following indicators are set for program SCB:

- Indicator {4-3} - set to the displacement relative to KODE (510) at which the statement begins
- KODE (933) - set to one
- Working register {5.} - set to one.

Before any exit, the output parameter from program SCA is set to one of the following values:

1 - continue execution in subsystem 2

2 - route system flow to subsystem 1, entry 1 for normal initialization

3 - route system flow to program LST for processing of LIST or EXPLAIN

4 - route system flow to subsystem 1, entry 2 for program control

5 - route system flow to subsystem 1, entry 2 for program control

6 - route system flow to subsystem 4 to output error message

7 - route system flow to subsystem 1, entry 3 for data area control.

When a label has been detected by program SCA, the first six characters are compared with entries in the following four tables, in the order listed:

- System label table {22.1}
- Variable table {1.2-1.5}
- Program call table {1.1.8}
- User program table {21.4}.

The labels in each table are packed two characters per word. The last three tables are searched only if they contain entries as indicated by the respective values of the variable count {15.}, the program count {1.1.1}, and the number of user programs on file {14.}. However, the user program table rather than the program call table is searched when the label to be matched is immediately preceded by one of the system labels: LIST, EDIT, or NAME. The first three tables are not searched if the label contains only one character. Both the system label table and the user program table are stored on file and read into core at the first occurrence of a label requiring a search of the particular table. The actions taken when a match to the label is found in one of the four tables and when the label does not occur in any of the tables is described in the following discussions.

4.4.2.1 System Label Table - Table 4-3 shows the contents of the system label table which is on the system control file {22.}. The first six entries are labels for which program SCA must take immediate action. The 200 numbers are the internal codes to be used for the label throughout the remainder of subsystem 2 and are 200 plus the interpreter

TABLE 4-3. SYSTEM LABEL TABLE

Label	Internal or Control Code	Label	Internal or Control Code
RESET		LE	218
SUPPRESS		SUB	243
END		INPUT	204
CARD		MIN	229
TO	212	MAX	230
ALL		TYPE	711
TYPEOUT	209	ABS	236
LIST	1001	TANH	237
EDIT	1002	SUM	238
NAME	1003	TAB	713
ARRAY	701	EXPLAIN	1004
IF	702	SUMF	232
THEN	703	MAGNITUDE	239
ELSE	704	PI	397
REPEAT	705	DEGREES	399
LAST	709	SIN	247
PAUSE	706	COS	248
EXIT	707	LN	234
SQ	708	EXP	249
INTERVALS	709	SQRT	250
GO	710	ATAN	235
DELETE	275	POW	258
SAVE	276	GO TO	212
LT	213	THRU	269
GT	214	SHIFT	223
EQ	215	PUNCH	712
NE	216	RANGE	701
GE	217		

operator code (see Table 2-4) for the label. The two 300 numbers are system constants. The 700 numbers are operators which require special action by program SCB, and the 1000 numbers are labels which require special but not immediate action by program SCA.

Upon recognition of one of the six labels which appear first in the table, the described actions are taken:

- RESET: Set the statement entry mode {16.} to one and the previous variable count {1.1.3} to zero; exit to subsystem 1, entry 1.

- SUPPRESS: Set the type of entry to RST {3.4} to four; exit to subsystem 1, entry 2.

- END: When entered in the conversational mode, set the type of entry to DLT {3.7-1} to two; exit to subsystem 1, entry 3.

  When entered in the SUPPRESS or EDIT mode, take the same actions as for SUPPRESS.

- CARD: Take the same actions as for SUPPRESS.

- TO: Replace with a blank the code for the label GO, which is in the output string; enter the code for TO in the output string; continue processing the statement.

- ALL: If preceded by the label LIST, set the type of entry to program LST {3.10-1}; exit to program LST.

  If preceded by the label EXPLAIN, set the output entry number {3.9-1} to 39; set the type of entry to program LST {3.10-1} to zero; exit to program LST.

If the label matched is not one of the first six entries in the system label table, the number provided in the table (column 4) is placed in the output string. If the system label is preceded by the operator EXPLAIN, program SCA sets the type of entry to program LST {3.10-1} to zero and the output entry number {3.9-1} to the position in the table (the row number) of the label just located and exits to program LST.

If the system label is TYPEOUT, program SCA places the TYPEOUT code in the output string and locates the first prime, which is the beginning delimiter for the message. Beginning with first character following the prime, the message is packed two characters per word and placed in the output string, beginning in the second word after the TYPEOUT operator. The packing is continued until the terminating prime is encountered, packing a blank in the last half word if required. A word count is stored in the output string immediately behind the TYPEOUT operator. The count is the number of words required for the packed form. Processing of the input statement is continued, starting with the first character after the closing prime.

4.4.2.2 Variable Table - Program SCA searches the labels currently entered in the variable table {1.2-1.5} to determine if the label is a variable name. The number of rows to be searched, beginning with row one, is specified by the current value of the variable count {15.}. When a match is found, the corresponding variable reference number is entered in the output string. The reference number is 400 plus the number of the row containing the variable name. If the variable is immediately preceded by the operator NAME, the packed variable name followed by a blank in AMTRAN character code is placed in the next four words of the output string following the variable reference number.

4.4.2.3 Program Call Table - The current value of the program count {1.1.2} determines the number of columns, beginning with column one, in the program call table {1.1.8} which are searched. The corresponding program reference number is placed in the output string upon a match. The reference number is 100 plus the number of the column containing the program name.

4.4.2.4 User Program Table - The number of rows in the user program table {21.4} searched, beginning with row one, is specified by the number of programs stored on file {14.}. When a match occurs and the label is not preceded by one of the special system labels, i.e., LIST, EDIT, or NAME, and the system is in the conversational or SUPPRESS mode, the program count {1.1.2} is incremented, the program name is entered into the corresponding column of the program call table {1.1.8}, and the program reference number which is 100 plus the program count is entered into the output string. The same actions are taken if the system is in the EDIT mode and the program name is not the name of the program being edited. If the names are the same, no action is taken, as if a match was not found. The names are the same if the row in the user program table is the same as the location in the table {3.1-1} set when the EDIT was initiated.

If the program name is preceded by one of the special system labels, the appropriate actions are taken:

- LIST
  - ▲ Set the type of entry to LST {3.10-1} to the length of the source form of the program. The length is obtained from column 6 of the program table.
  - ▲ Set the LIST control {3.9-2} to the record number on the user program file {22.} at which the internal form of the program begins. The record number is in column 4 of the program table.
  - ▲ Exit, routing system flow to program LST.
- EDIT
  - ▲ Set the program table location {3.1-1} to the row in the program table containing the program name.
  - ▲ Set the FORTRAN record control {7.} to the record on the user program file {22.} which contains the program statement count. The record number is in column 5 of the program table.
  - ▲ Exit, routing system flow to subsystem 1, entry 2.

- NAME (valid only in EDIT mode)

  - ▲ Place a zero, the three word program name, and an AMTRAN character coded blank in the next five words of the output string.

  - ▲ Continue processing the input statement.

4.4.2.5 Label Not Appearing in Tables - When a label has not been matched with an entry in one of the four tables, it is entered in the variable table {1.2-1.5}. The variable count {15.} is incremented, the label is entered in the corresponding row of the variable table, and the variable reference number (400 plus the variable count) is placed in the output string.

4.4.3 Program SCB

Program SCB completes the conversion, initiated by program SCA, of an AMTRAN source statement from a string of characters to a string of internal codes. Program SCA has already converted label strings to internal or temporary codes. Program SCB scans the input statement and converts the remaining digital and special characters to internal codes; performs special formatting for the operators ARRAY, PAUSE, REPEAT, EXIT, SQ, INTERVALS, TYPE, PUNCH and TAB, and for the IF statement; and recognizes the data control commands DELETE and SAVE upon completion of the statement conversion.

The partially converted statement is input to program SCB in state 2 of the statement conversion area {1.2-1.1}. The displacement relative to KODE (510) at which the statement begins is provided in the working register {4-3}. For convenience, two indicators, KODE (933) and the working register {5.}, used by program SCB have been initialized by program SCA to one.

The fully converted statement is output in state 3 of the statement conversion area {1.2-1.1}. In addition, the program SCB outputs a parameter which is set to indicate one of the following:

1 - Continue statement translation.

2 - Route system flow to subsystem 1, entry 3 for processing of a DELETE or SAVE statement.

3 - Route system flow to subsystem 4 for output of an error message.

The input string is a mixture of the following AMTRAN character, internal, and special codes:

- All character codes listed in Table 4-2, except those for the alphabetic characters
- Program and variable references (numbers in the ranges 101 through 110 and 400 through 429, respectively)
- The operator and special codes listed in Table 4-3 (numbers in the ranges 201 through 269 and 701 through 713, respectively).

The ouput string consists of numbers in the range 99 through 494 which have the following designations:

99 : end of statement
101 - 110 : program references
201 - 269 : operators and delimiters
301 - 355, 386 - 399 : constant references
400 - 430 : variable references
450 - 494 : statement references.

The program and variable reference numbers are assigned by program SCA and explained in the description of that program (see Section 4.4.2). Table 4-4 lists the operators and delimiters and their corresponding internal codes. The constant and statement reference numbers are generated by program SCB and are discussed in the following program description.

TABLE 4-4. OPERATOR AND DELIMITER CODES OUTPUT BY PROGRAM SCB

Code	Operator or Delimiter	Code	Operator or Delimiter
201	EXIT	237	TANH
203	PAUSE	238	SUM
204	INPUT	239	MAGNITUDE
205	TYPE	241	THEN
206	PUNCH	242	ELSE
207	TAB	243	SUB
209	TYPEOUT	247	SIN
212	GOTO	248	COS
213	LT	249	EXP
214	GT	250	SQRT
215	EQ	251	NEGATION
216	NE	258	EXPONENTIATION
217	GE	259	*
218	LE	260	/
222	ARRAY	261	+
223	SHIFT	262	- (SUBTRACT)
229	MIN	263	&
230	MAX	264	=
231	INTERVALS	265	(
232	SUMF	266	)
234	LN	268	,
235	ATAN	269	THRU
236	ABS	275	DELETE
		276	SAVE

The input elements can be grouped into the following types:

Type	Elements
1	Digits zero through nine
2	Operator TYPEOUT
3	Program references, variable references and all operators not of types 2, 4, and 5
4	Relational operators
5	Operators which require special formatting (those numbers in the range 701 through 713)
6	Special characters.

The processing of the input elements is shown in Figure 4-8 and described in the following discussions.

4.4.3.1 Type 1 - When a digit occurs in the input string, the last element entered into the output string is checked. If the element is the GO TO operator (code 212), the digit is part of a statement number which can appear in the input string in any of the forms listed below where d represents a digit and is set to one in the examples in parentheses.

d	(1)	dd.d	(11.1)
d.	(1.)	dd.dd	(11.11)
dd	(11)	.d	(.1)
dd.	(11.)	.dd	(.11)

The statement number is replaced by a statement reference which is 449 plus the number of the row in the statement index table {1.2-1.3} which contains that statement number. In the conversational mode and in the SUPPRESS mode with typewriter input, the statement number is also the row number. In the EDIT mode and when input is from cards, the table must be searched to determine the row number.

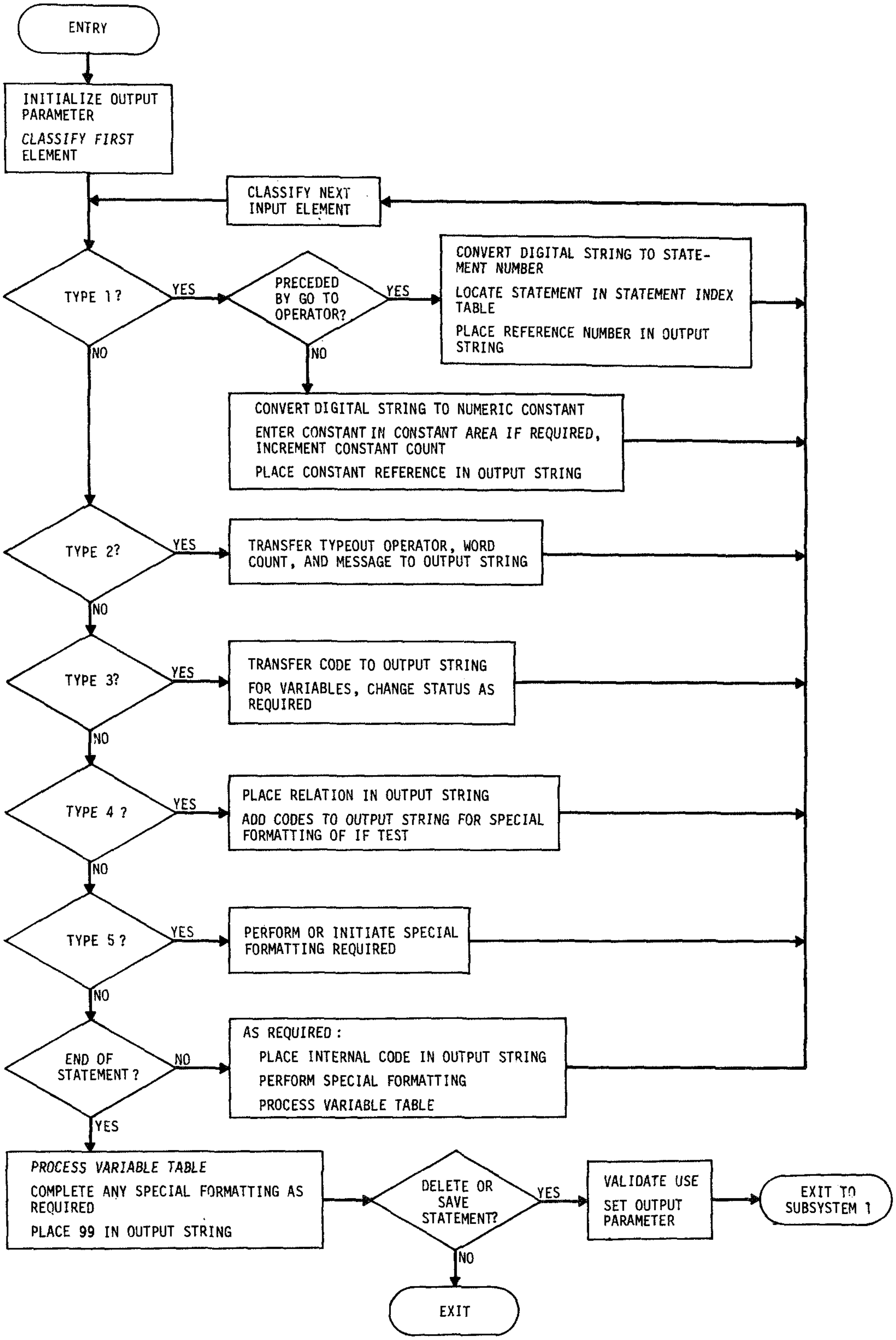

ENTRY
INITIALIZE OUTPUT PARAMETER
CLASSIFY FIRST ELEMENT
CLASSIFY NEXT INPUT ELEMENT
TYPE 1?
YES
NO
PRECEDED BY GO TO OPERATOR?
YES
NO
CONVERT DIGITAL STRING TO STATEMENT NUMBER
LOCATE STATEMENT IN STATEMENT INDEX TABLE
PLACE REFERENCE NUMBER IN OUTPUT STRING
CONVERT DIGITAL STRING TO NUMERIC CONSTANT
ENTER CONSTANT IN CONSTANT AREA IF REQUIRED, INCREMENT CONSTANT COUNT
PLACE CONSTANT REFERENCE IN OUTPUT STRING
TYPE 2?
YES
NO
TRANSFER TYPEOUT OPERATOR, WORD COUNT, AND MESSAGE TO OUTPUT STRING
TYPE 3?
YES
NO
TRANSFER CODE TO OUTPUT STRING
FOR VARIABLES, CHANGE STATUS AS REQUIRED
TYPE 4 ?
YES
NO
PLACE RELATION IN OUTPUT STRING
ADD CODES TO OUTPUT STRING FOR SPECIAL FORMATTING OF IF TEST
TYPE 5 ?
YES
NO
PERFORM OR INITIATE SPECIAL FORMATTING REQUIRED
END OF STATEMENT ?
NO
YES
AS REQUIRED :
PLACE INTERNAL CODE IN OUTPUT STRING
PERFORM SPECIAL FORMATTING
PROCESS VARIABLE TABLE
PROCESS VARIABLE TABLE
COMPLETE ANY SPECIAL FORMATTING AS REQUIRED
PLACE 99 IN OUTPUT STRING
DELETE OR SAVE STATEMENT?
YES
NO
VALIDATE USE
SET OUTPUT PARAMETER
EXIT TO SUBSYSTEM 1
EXIT

FIGURE 4-8. PROGRAM SCB

When the first digit in a string is not preceded by the GO TO operator, the digital string, which may contain a decimal point, is converted to a constant reference. The reference is in either the range 301 through 354 or the range 386 through 399 and refers to either a user constant or system constant, respectively. The digital string is converted to the corresponding floating point representation, compared with the system constants {1. 3. 2}, and replaced by the corresponding reference, if possible. Table 4-5 lists the system constants and reference numbers. If it is not a system constant, the number is compared with any constants appearing in the user constants area {1. 1. 7}. The number of entries is specified by the value of the constant count {11. }. If the number does appear, the corresponding constant reference is 300 plus the row in which the number is entered in the constant area. When the number does not already appear, it is entered in the next available row in the constant area, the constant count is incremented, and the corresponding constant reference is assigned. All constants are entered as positive numbers. For negative numbers, the minus sign has already been processed and the appropriate operator placed in the output string (see Section 4. 4. 3. 6).

4. 4. 3. 2 Type 2 - In the input string, the operator TYPEOUT is immediately followed by a word count specifying the number of sequential words which contain the user message to be output. The operator, word count, and message are transferred directly to the output string.

4. 4. 3. 3 Type 3 - The elements of this type are transferred directly to the output string. For a variable reference, the corresponding column 4 and column 5 entries in the variable table {1. 2-1. 5} are summed. If the sum is positive, no action is taken. When the sum is negative, the entry in column 4 is set to zero. If the sum is

TABLE 4-5. SYSTEM CONSTANTS AND REFERENCE NUMBERS

Constant	Reference
0.0	386
1.0	387
2.0	388
3.0	389
4.0	390
5.0	391
6.0	392
7.0	393
8.0	394
9.0	395
10.0	396
3.1415927	397
57.2958	398
0.0174533	399

zero, the element preceding the variable reference in the output string is checked. If the element is INPUT (code 204), the columns 4 and 5 entries are set to one; otherwise, the column 5 entry is set to minus two.

4.4.3.4 Type 4 - The processing of a relational operator is part of the special formatting performed for an IF test (see Section 4.4.3.5). Program SCB places the following elements in the output string in the order listed: right parenthesis, subtract, left parenthesis. The relational operator is inserted in the output string before the first relational argument (see Section 4.4.3.5).

4.4.3.5 Type 5 - The operators of this type, with the exception of INTERVALS, require the addition of codes to the output string to simplify the next phase of the translation process. For INTERVALS, program SCB determines whether the label is used as a subscript argument or as a function operator by checking the last element in the output string. If the last element is either SUB or THRU (codes 243 and 269, respectively), subscripting is the use and a 400 is placed in the output string. Otherwise, INTERVALS as a function operator (code 231) is placed in the output string.

The formatting for PAUSE, EXIT, and SQ is done immediately. The operator codes for PAUSE and EXIT (see Table 4-4) are placed in the output string and followed by the dummy variable reference number 400. Operator SQ is replaced by the codes for the string: POW 2.0.

Operators ARRAY, TYPE, PUNCH, and TAB cannot be totally reformatted when first encountered. For each, the formatting consists of enclosing the associated arguments in parentheses. The left parenthesis is placed directly behind the operator code in the output string. The matching right parenthesis is placed in the output string when one of the following occurs in the input string:

- Semicolon
- THEN
- ELSE
- End of statement

or when a comma occurs and the following conditions have been met for the indicated operator:

- ARRAY - three arguments have been processed.
- TYPE, TAB, and PUNCH - the secondary parentheses count is balanced (see Section 4.4.3.6).

The IF test, whether appearing as a statement or as an embedded substructure, is reorganized from the general input form

$$\text{IF } a_1 \text{ relation } a_2 \text{ THEN } s_1, \ldots, s_n$$

$$\text{ELSE } t_1, \ldots t_m$$

to the output form

$$\text{relation } ((a_1) - (a_2)), \text{ THEN } (s_1, \ldots, s_n),$$

$$\text{ELSE } (t_1, \ldots, t_m)$$

where

relation	- one of the relational operators
a_1 and a_2	- the arguments for the relation
$s_1, \ldots, s_n$ and $t_1, \ldots, t_m$	- the substatements comprising the THEN and ELSE clauses, respectively; n and m may be one or greater.

Program SCB keeps count of the number of IF tests which are currently being reformatted in order to correctly delimit embedded IF tests. When an IF operator is encountered, program SCB counts the IF test, leaves a one-word gap in the output string to be filled in with the relational operator (see Section 4.4.3.4), and places two left parentheses in the output string.

To keep track of THEN/ELSE pairs and to handle any ELSE clauses which do not appear in the input statement, the program maintains a control table. When a THEN occurs, the THEN is entered in the control table. If the last element in the output string is a comma, it is deleted. To the output string are added any right parentheses required to complete an ARRAY sequence and/or a TYPE, PUNCH, or TAB sequence. The following sequence is then placed in the output string:

- Right parenthesis
- Right parenthesis
- Comma
- THEN
- Left parenthesis

When an ELSE occurs and the last entry in the control table is an ELSE:

- Remove the ELSE and the matching THEN from the table.
- Decrement the IF test count.
- Place a right parenthesis in the output string.

This sequence is continued until a THEN is the last entry in the control table. The tasks listed for the THEN operator are then performed using an ELSE instead of a THEN. The formatting of the IF test is completed when a semicolon or an end of statement occurs (see Section 4.4.3.6).

The REPEAT statement is input in the form

$$\text{REPEAT } n,\ s_1, \ldots, s_m$$

where n is the number of times the substatements $s_1, \ldots, s_m$ are to be executed. Program SCB places the statement in the form

$$R = 0,\ R = R + 1,\ \text{LE } (R - (n)),$$

$$\text{THEN } (s_1, \ldots, s_m,\ \text{GO TO } 355)$$

where R refers to a special temporary variable (code 430) which is used only in REPEAT statements and for counting the execution loops;

zero (0) and one (1) are system constants (codes 386 and 387, respectively); LE is the relational operator less than; and the number 355 is a dummy reference number used only in REPEAT. When the REPEAT operator occurs in the input string, the codes for the following sequence are read from the system control file {22.2} and placed in the output string:

- Repeat temporary
- Equal
- System constant 0.0
- Comma
- Repeat temporary
- Equal
- Repeat temporary
- Plus
- System constant 1.0
- Comma
- LE
- Left parenthesis
- Repeat temporary
- Subtract
- Left parenthesis.

The formatting is continued when the first comma occurs and is completed on the end of statement (see Section 4.4.3.6).

4.4.3.6 Type 6 - For the following special characters, program SCB takes the action indicated:

- Blank, period, carriage return - None : skip to next input element.

- Plus sign, slash, ampersand - Place the operator code (see Table 4-4) in the output string.

- Asterisk - If the next element in the input string is an asterisk, place the exponentiation operator in the output string and skip the second asterisk; otherwise, place the multiplication operator in the output string.

- Minus sign - If the last element entered in the output string is a right parenthesis, THRU, constant reference, or variable reference, place the subtract operator in the output string; otherwise, place the negation operator in the output string.

- Equal sign - Make the first undefined user variable entered in the variable table {1.2-1.5} defined: set to one the entry in column 4 which corresponds to the first negative entry in column 5; and place the equal in the output string.

Left and right parentheses are always entered directly into the output string. Throughout the processing of a statement, program SCB checks for parentheses pairs in the input statement to ensure that the parentheses are balanced. In addition, to ensure that the parentheses added to a statement by SCB (see Section 4.4.3.5) do not change the meaning of the statement, a secondary parentheses count is maintained. The count becomes active on a left parenthesis in the input statement when the last entry in the output string is either a program reference or the operator SHIFT, or when the second to last entry is one of the operators TYPE, PUNCH, or TAB. The count is inactive when the parentheses become balanced within that portion of the string being currently processed.

The comma is a key delimiter in the reformatting of the REPEAT statement and the operators ARRAY, TYPE, PUNCH, and TAB (see Section 4.4.3.5). When a comma in the input string is the first comma to occur after a REPEAT operator, the following sequence is placed in the output string:

- Right parenthesis
- Right parenthesis
- Comma
- THEN
- Left parenthesis.

Also, the status of all current user variables is finalized in the variable table {1.2-1.5} by setting all negative entries in column 5 to one. If the comma is not the first comma after a REPEAT, the comma is placed in the output string after adding any right parentheses required for the ARRAY, TYPE, PUNCH, and TAB operators. The conditions for inserting the right parentheses are detailed in Section 4.4.3.5.

Upon a semicolon, any right parentheses required for these four operators are inserted and an additional right parenthesis is placed in the output string. If the last entry in the control table (see Section 4.4.3.5) is an ELSE, the last THEN and ELSE are removed from the table. If the last entry is a THEN, the THEN is removed from the table and the following sequence is entered in the output string:

- Comma
- ELSE

In either case, the IF count is decremented and the next input element is checked. If the element is not a semicolon, the actions described for a comma are taken.

When the end of statement character occurs, program SCB first verifies that the total parentheses are balanced. The program then completes the special formatting for the Type 5 elements. One right parenthesis is added to the output string for each of the following operators which are in the process of being formatted:

- ARRAY
- TYPE, PUNCH, or TYPE
- IF (may be more than one).

To complete a REPEAT statement, the following sequence is added to the output string:

- Comma
- GO TO
- The dummy statement reference number 355
- Right parenthesis.

In all cases, the status of all variables is finalized by making all entries in column 5 of the variable table {1.2-1.5} one, and the termination code 99 is placed in the output string. If the first element in the string is a DELETE or SAVE, the output parameter is set to two.

4.5 SUBSYSTEM 3

4.5.1 Program STK

Program STK, shown in Figure 4-9, converts the internal statement string generated by subsystem 2 to the equivalent post-fix Polish stack. The input string is provided in state 3 of the statement conversion area {1.2-1.1}. The stack is output in state 4 of the statement conversion area.

The input string and output stack consist of numbers in the range 99 through 494 which have the following designations:

99 : end of statement
101 - 110 : program references
201 - 273 : operators and delimiters
301 - 355, 386 - 399 : constant references
400 - 430 : variable references
450 - 494 : statement references.

The stack generated by program STK is a string in which the priority of operations is provided by the operational sequence from left to right. The operations of highest priority within a term are leftmost in the string. For each operation, the associated operands immediately precede the operation. The task of program STK is to scan the input string and reorder it, based on operation priorities, to generate the output stack. The reordering is accomplished using the output stack and a delimiter list which functions as a last-in, first-out stack.

Since only operations are reordered, statement, constant, and variable reference numbers are transferred to the stack as they occur in the input string. To remove implied multiplication, the multiply operator is inserted into the string after a constant or variable reference when the next input element is one of the following:

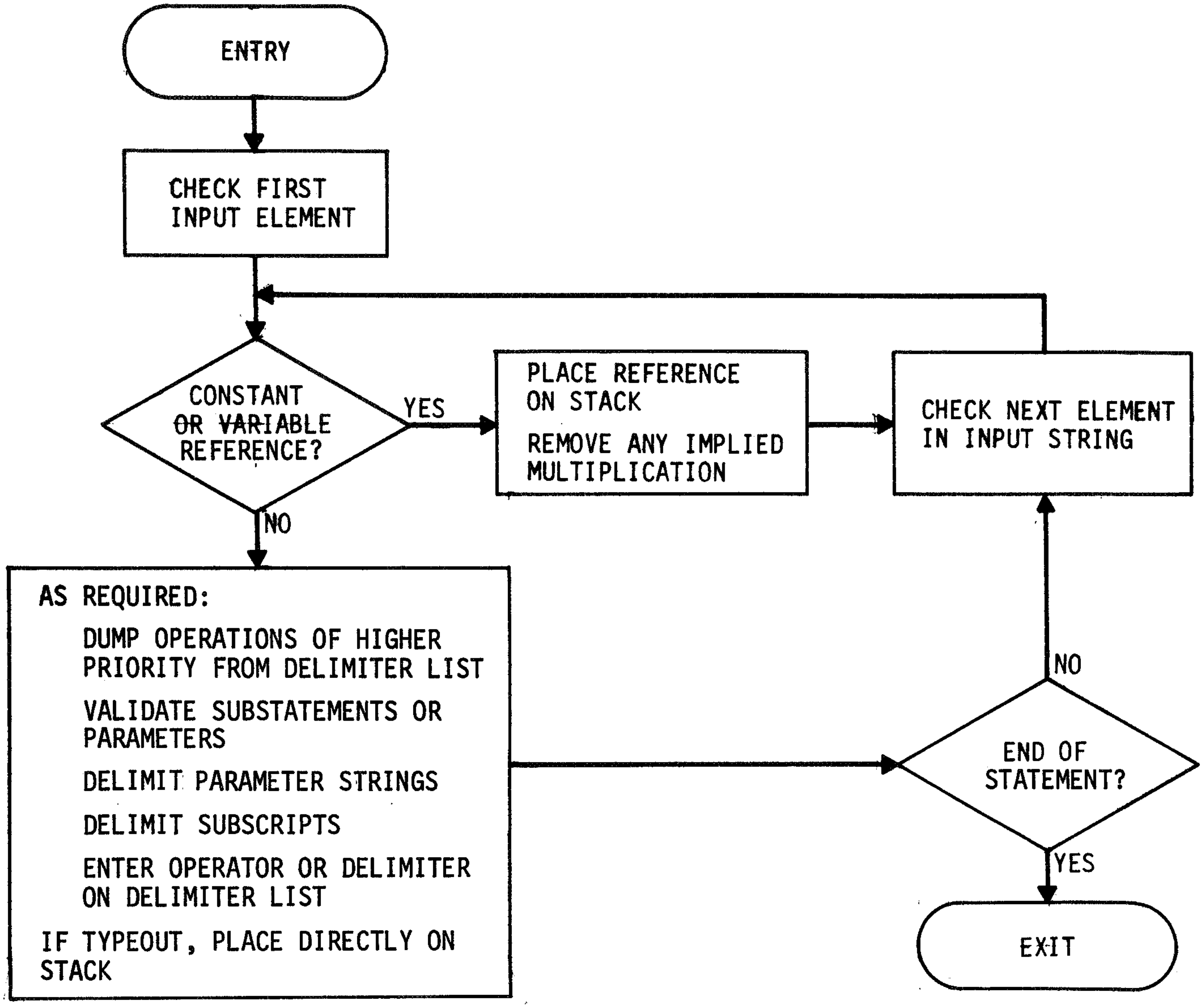

ENTRY
CHECK FIRST INPUT ELEMENT
CONSTANT OR VARIABLE REFERENCE?
YES
PLACE REFERENCE ON STACK
REMOVE ANY IMPLIED MULTIPLICATION
CHECK NEXT ELEMENT IN INPUT STRING
NO
AS REQUIRED:
DUMP OPERATIONS OF HIGHER PRIORITY FROM DELIMITER LIST
VALIDATE SUBSTATEMENTS OR PARAMETERS
DELIMIT PARAMETER STRINGS
DELIMIT SUBSCRIPTS
ENTER OPERATOR OR DELIMITER ON DELIMITER LIST
IF TYPEOUT, PLACE DIRECTLY ON STACK
NO
END OF STATEMENT?
YES
EXIT

FIGURE 4-9. PROGRAM STK

- Constant reference
- Variable reference
- Left parenthesis
- Program reference
- The operators listed in Table 4-4 with codes less than 258.

Within arithmetic expressions, the order of computation is determined by the relative priorities of the operations - the operation of highest priority being done first. The priority of operations is:

1. Operations within parentheses
2. Functions (such as user defined functions, SIN, ARRAY, etc.)
3. Exponentiation and negation
4. Multiplication and division
5. Addition and subtraction
6. Relational operations
7. Concatenation.

Program names have been replaced by program references which are in the range 101 to 110, and operations and delimiters have the 200 range numbers listed in Table 4-4. Thus, the numbers assigned closely parallel the inverse order of priority. Program STK uses this characteristic in generating the stack. Operators are placed in a delimiter list after first "dumping" any operation of higher priority from the list to the stack. Except for priority levels 2 and 3, operations of the same priority are also dumped.

Parameter strings are placed in the stack preceding the associated operator. The parameters are blocked off in the stack by the addition of matching delimiter pairs. The left delimiter has code 273 and the right delimiter has code 272. The left delimiter is placed in the stack and the right delimiter is placed in the delimiter list behind the operator when a program reference or one of the following operators occurs:

- ARRAY
- TYPE
- TAB
- PUNCH
- SHIFT.

When the parameter string is enclosed in parentheses, the left parenthesis is placed in the delimiter list between the operator and the right parameter string delimiter so that the matching right parenthesis will eventually dump the right delimiter. The parentheses do not get placed in the stack.

Subscripts are similarly bracketed. However, the subscript modification appears behind the variable reference rather than before it in the output stack. The left subscript delimiter (code 271) is placed in the stack when operator SUB occurs and replaces the SUB. The right subscript delimiter has code 270.

The operation priority list presented previously is not sufficient to handle all of the input operators and delimiters appearing in Table 4-4. In the following discussions, those operators and delimiters which obviously do not fall into the priority levels two through seven will be explained.

The label delimiters THEN, ELSE, GO TO, INPUT, TYPE, PUNCH, TAB, PAUSE, and EXIT, and the noncomputational commands are treated as having the priority of a functional operator (level 2). The functional operators cannot dump anything in the delimiter list and are simply added to the list.

The operator TYPEOUT is an exception in program STK. The operator TYPEOUT is followed in the input string by a word count which specifies the number of subsequent words which contain the user message to be output. The operator, word count, and message are transferred in order directly to the output stack.

A left parenthesis is entered directly on the list. A right parenthesis will dump all entries on the list until either the matching left parenthesis or a subscript delimiter is found. In the first case, the left parenthesis is deleted from the list. In the second case, the right subscript delimiter is placed in the stack. Parentheses are never entered in the stack.

The only entry an equal can dump from the delimiter list is a right subscript bracket. The equal is always placed in the delimiter list.

The operator THRU dumps all entries in the delimiter list until a subscript delimiter occurs. The THRU is then discarded: the THRU is not placed on the stack or in the list.

The comma causes everything on the delimiter list to be dumped until either a left parenthesis or a parameter string delimiter is encountered. If the parameter string delimiter is preceded by a left parenthesis in the list, the parameter string delimiter is dumped to the stack and the dumping continues. The comma is never placed on the delimiter list. It is placed in the stack after all dumping has been done if the comma does not occur inside parameter string delimiters.

The end of statement (code 99) dumps everything from the delimiter list to the stack. The end of statement is then placed on the stack.

A further task which program STK performs is a validity check on substatements and parameter strings. The last element entered in the stack is checked

- After a comma is processed
- After everything has been placed in the stack on an end of statement
- Whenever a parameter string right delimiter is the next element to be placed on the stack.

The element is checked to ensure that only a program reference or one of the following appears as a substatement:

- EXIT
- PAUSE
- INPUT
- TYPE
- PUNCH
- TAB
- TYPEOUT
- GO TO
- A relational operation
- A THEN or ELSE clause
- Assignment.

A parameter is valid if the last element is not one of the operations listed above.

4.5.2 Program CDR

Program CDR, shown in Figure 4-10, generates interpreter instructions from the post-fix Polish stack. The stack is input as a string in state 4 of the statement conversion area {1.2-1.1}. The program CDR processes the string from left to right, generating the instructions for each operation as it appears. The generated instructions are entered in order into the interpreter instructions area {1.1.5}, beginning at the location specified by the value of the last previous instruction pointer {13.}. Based on this pointer and on the number of instructions generated, the program sets the last instruction pointer {6.} before exiting. Also, the program places column 2 of the current entry in the statement index table {1.2-1.3} in form 2. The current entry is specified by the value of the statement count {18.}.

The input string consists of numbers in the range 99 through 494 which have the following designations:

99: end of statement
101 - 110 : program references
201 - 273 : operators and delimiters
301 - 355, 386 - 399 : constant references
400 - 430 : variable references
450 - 494 : statement references.

The input string can contain the codes listed in Table 4-4 for operators and delimiters (except for SAVE and DELETE) and the codes listed below for special delimiters.

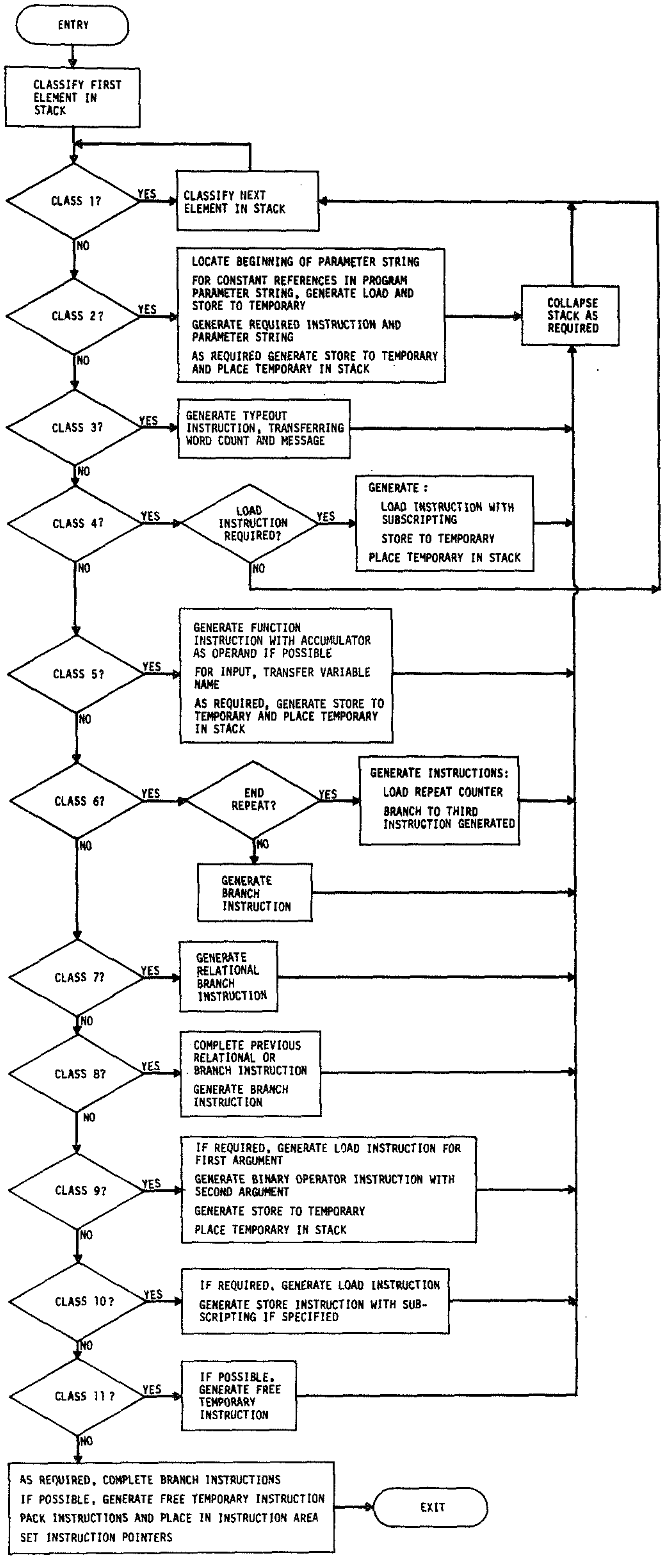

ENTRY
CLASSIFY FIRST ELEMENT IN STACK
CLASS 1?
YES
CLASSIFY NEXT ELEMENT IN STACK
NO
CLASS 2?
YES
LOCATE BEGINNING OF PARAMETER STRING
FOR CONSTANT REFERENCES IN PROGRAM PARAMETER STRING, GENERATE LOAD AND STORE TO TEMPORARY
GENERATE REQUIRED INSTRUCTION AND PARAMETER STRING
AS REQUIRED GENERATE STORE TO TEMPORARY AND PLACE TEMPORARY IN STACK
COLLAPSE STACK AS REQUIRED
NO
CLASS 3?
YES
GENERATE TYPEOUT INSTRUCTION, TRANSFERRING WORD COUNT AND MESSAGE
NO
CLASS 4?
YES
LOAD INSTRUCTION REQUIRED?
YES
GENERATE :
LOAD INSTRUCTION WITH SUBSCRIPTING
STORE TO TEMPORARY
PLACE TEMPORARY IN STACK
NO
NO
CLASS 5?
YES
GENERATE FUNCTION INSTRUCTION WITH ACCUMULATOR AS OPERAND IF POSSIBLE
FOR INPUT, TRANSFER VARIABLE NAME
AS REQUIRED, GENERATE STORE TO TEMPORARY AND PLACE TEMPORARY IN STACK
NO
CLASS 6?
YES
END REPEAT?
YES
GENERATE INSTRUCTIONS:
LOAD REPEAT COUNTER
BRANCH TO THIRD INSTRUCTION GENERATED
NO
GENERATE BRANCH INSTRUCTION
NO
CLASS 7?
YES
GENERATE RELATIONAL BRANCH INSTRUCTION
NO
CLASS 8?
YES
COMPLETE PREVIOUS RELATIONAL OR BRANCH INSTRUCTION
GENERATE BRANCH INSTRUCTION
NO
CLASS 9?
YES
IF REQUIRED, GENERATE LOAD INSTRUCTION FOR FIRST ARGUMENT
GENERATE BINARY OPERATOR INSTRUCTION WITH SECOND ARGUMENT
GENERATE STORE TO TEMPORARY
PLACE TEMPORARY IN STACK
NO
CLASS 10?
YES
IF REQUIRED, GENERATE LOAD INSTRUCTION
GENERATE STORE INSTRUCTION WITH SUB-SCRIPTING IF SPECIFIED
NO
CLASS 11 ?
YES
IF POSSIBLE, GENERATE FREE TEMPORARY INSTRUCTION
NO
AS REQUIRED, COMPLETE BRANCH INSTRUCTIONS
IF POSSIBLE, GENERATE FREE TEMPORARY INSTRUCTION
PACK INSTRUCTIONS AND PLACE IN INSTRUCTION AREA
SET INSTRUCTION POINTERS
EXIT

FIGURE 4-10. PROGRAM CDR

Code	Delimiter
270	right subscript
271	left subscript
272	right parameter string
273	left parameter string

The elements of the input string may be grouped into classes which reflect the functions required to generate the equivalent interpreter instructions. The twelve classes are listed below.

Class	Elements
1	Constant references, variable references, left and right parameter string delimiters, and the left subscript delimiter
2	The operations which have parameter strings enclosed in delimiters: ● Program references ● TYPE ● TAB ● PUNCH ● ARRAY ● SHIFT
3	The operator TYPEOUT
4	Right subscript delimiter (code 270)
5	The following operators which have only one operand or are preceded by a dummy operand: EXIT, PAUSE, INPUT, MIN, MAX, INTERVALS, SUMF, LN, ATAN, ABS, TANH, SUM, MAGNITUDE, SIN, COS, EXP, SQRT, and negation
6	The operator GO TO
7	The relational operators
8	THEN and ELSE
9	Binary arithmetic operators
10	Equal
11	Comma
12	End of statement

The program CDR moves along the string, generating the instructions required to perform each operation and to store any results. The result replaces the operation and operands in the stack. For an operation which does not generate a result, the operation and associated operands are removed from the stack. This process continues until the stack is completely processed.

The format for all interpreter instructions is described in Table 2-4. Constant, program, and variable references are input to program CDR in the final form required for the instructions. However, operations are in the range 201 through 263. Before the instructions are finalized, 200 is subtracted from the operation codes. The delimiters do not appear in instructions.

In addition to generating the instructions which correspond directly to the operations in the input string, program CDR generates load and store interpreter instructions and assigns temporary variables. The internal codes which refer to temporary variables are variable reference numbers in the range 431 through 450. They are assigned in order, beginning with 431, and reassigned for reuse as soon as possible. To perform this assignment, program CDR maintains a temporary count which will be referred to in the discussions on the various input element classes. The count is initialized to 430 and is incremented by one before each assignment. The count is decremented as a temporary becomes available for further use; however, the count is not allowed to drop below 430.

4.5.2.1 Class 1 - The constant and variable references and delimiters comprising this class are not processed until the associated operand occurs.

4.5.2.2 Class 2 - To process an operation which is preceded by a parameter string, the input stack is searched backwards from the operation until the left parameter string delimiter is located. If the operation is a program reference, for each element in the string which is a constant reference, the program generates a load and a store instruction (see Table 2-4) with the constant reference and a temporary variable for the respective operands. The constant reference in the parameter string is replaced with the temporary variable reference. When the left parameter string delimiter is located, the program generates the appropriate instructions, which are described in Table 2-4. The parameters are placed in subsequent words in the order in which they occur in the string from left to right. As each temporary variable reference is transferred, the temporary count is decremented.

When the parameter transfer is completed and either a TYPE, TAB, or PUNCH is being processed, the operator and its parameter or parameters are removed from the stack: the stack is compressed removing the elements, beginning with the left parameter string delimiter and ending inclusively with the operator.

For the operators ARRAY and SHIFT and a program call, a store to temporary instruction is generated after the last parameter word. The temporary reference replaces the operation or program reference, associated parameters, and the two parameter string delimiters in the stack.

4.5.2.3 <u>Class 3</u> - The operator TYPEOUT is followed in the stack by the word count which specifies the number of subsequent words in the stack occupied by the message to be output. The TYPEOUT instruction is generated, leaving the message words in order. The stack is compressed, removing the operator, word count, and message.

4.5.2.4 <u>Class 4</u> - For a right subscript delimiter, program CDR determines whether a variable is being subscripted to the left of an equal sign. If the delimiter is not beyond the fifth entry in the stack and if the second entry is a left subscript delimiter, the variable is to the left of an equal sign. In this case, the delimiter is not processed.

Otherwise, program CDR generates a load instruction for the variable whose reference number immediately precedes the left subscript delimiter. The subscript or two subscripts are transferred to the subsequent instruction word or words. For each subscript which is a temporary reference, the temporary count is decremented. The program then generates a store to temporary instruction and this temporary reference replaces the variable reference and parameter string with delimiters in the stack.

4.5.2.5 Class 5 - To process the functions which have a single operand, program CDR checks the last instruction generated. If the instruction is a store and the store operand is the same as the function operand but is not a temporary reference, the instruction for the function is generated with the system accumulator as the operand. If the matched operand is a temporary reference, the store to temporary instruction is deleted before the instruction using the system accumulator is generated. In all other cases, the operand preceding the function in the stack is used to generate the instruction.

For the PAUSE and EXIT instructions, the operator and dummy operand are removed from the stack. For the operator INPUT, the program locates the variable name in the variable table {1.2-1.5} and transfers the name to the three words following the INPUT instruction. The operator and variable reference are then removed from the stack.

For the remaining operators of class 5, a store to temporary instruction is generated after the function instruction. The temporary reference replaces the operator and operand in the stack.

4.5.2.6 Class 6 - In the stack, the GO TO operator is immediately preceded by a statement reference. The reference is 449 plus the number of the row in the statement index table {1.2-1.3} which contains the storage information for the statement to which a branch is made. If the row number is less than or equal to the statement count {18.}, program CDR determines the beginning location in the interpreter instructions area {1.1.5} of the statement. This information is obtained from column 2 which is in form 2. From this location, the displacement for the GO TO instruction is calculated and the instruction is generated. If the row number is greater than the statement count {18.} (this is acceptable only in the SUPPRESS and EDIT modes), the GO TO instruction is generated using the statement reference as a temporary operand.

The GO TO operator may be preceded in the input string by the dummy statement number 355. The code signals the end of a REPEAT statement (see Section 4.4.3.5). Program CDR generates a load instruction with the repeat counter (code 430) as the operand. The program then fills in any incomplete branches required for IF tests (see Section 4.5.2.8), making them branch to this load instruction. A GO TO is then generated to branch to the third word of the instruction sequence generated for the current statement.

After a branch instruction has been generated, the GO TO operator and the single operand are deleted from the stack.

4.5.2.7 Class 7 - For a relational operator, the corresponding instruction is generated without an operand. The operand is filled in when the delimiter THEN occurs in the stack. The relational operator and the preceding code are removed from the stack.

4.5.2.8 Class 8 - The THEN and ELSE are used in the stack as delimiters for THEN and ELSE clauses, respectively. They appear at the end of the clauses. When a THEN occurs, a branch instruction is generated with a null operand, to be changed when an ELSE occurs. The last relational instruction (see Section 4.5.2.7) is filled in with an operand based on the displacement from the relational instruction to the next instruction after the added branch.

When an ELSE occurs, a nonoperative branch instruction is generated. The branch is nonoperative in that the displacement used in the operand is zero. The last branch generated for a THEN is altered to cause a branch to the nonoperative instruction. After the instruction generation and modification have been completed, the THEN or ELSE is removed from the stack.

4.5.2.9 Class 9 - In processing a binary operator, program CDR examines the last instruction to determine whether or not a load instruction is required for the first operand. If the last instruction is a store, the operand is compared with the first operand of the binary operator. (The first operand is the leftmost of the two operands preceding the binary operator in the stack.) If the operands are the same, the load is not required; and, if the operand is a temporary reference, the store instruction is deleted. In all other cases, a load instruction is generated to load the first operand. If the argument is a temporary reference, the load becomes a load/free instruction. The instruction is generated for the binary operator, with the second argument as the operand. If either of the two operands is a temporary reference, the temporary count is decremented. A store to temporary instruction is generated. The temporary reference replaces the binary operator and two operands in the stack.

4.5.2.10 Class 10 - To process the equal, program CDR generates a load instruction for the operand immediately preceding the equal unless the last instruction generated is a store for the same operand. If this previously generated store is a store to temporary, the instruction is deleted.

The store instruction is generated which may or may not require subscripting. If subscripting is required, the subscript or subscripts will be enclosed in subscript delimiters (see Section 4.5.1) and will separate the two operands in the stack with which the equal is associated.

The store instruction operand is the first in the stack (the one farthest away from the equal). The two operands, the subscript string, if present, and the equal are removed from the stack.

4.5.2.11 Class 11 - The comma separates substatements in the stack and is used to free temporary variables. When a comma occurs, all instructions generated since either the last occurrence of a comma or the beginning of the statement are searched for temporary references. For each reference found, column 4 of the corresponding row (referencing is identical to variable referencing) in the variable table {1.2-1.5} is set to one. If any temporary variable references occur in the sequence of instructions, program CDR generates a free temporary instruction. The instruction has the first temporary (code 431) as its single argument, if only this temporary was allocated. Otherwise, the instruction has two operands. The first argument is the highest temporary reference number occurring in the instruction sequence. The second operand is always 431. The comma and anything preceding it are removed from the stack.

4.5.2.12 Class 12 - On the end of statement, any branches which have not been completed (see Section 4.5.2.8) are made to branch to the next instruction to be generated. A free temporary instruction is generated, if required (as explained for the class 11 element). An exit is then made from program CDR.

4.6 SUBSYSTEM 4

4.6.1 Data Referencing System

Most of the interpreter operators (see Table 2-4) may have operands which are constant or variable references or a reference to the system accumulator. Since different routines execute these operators, the mechanism for locating data will be discussed in this section.

A constant reference is a 300 number. If the number is in the range 301 to 354, it refers to a user constant stored in the constant area of the user program being executed (see Figure 2-1). The rightmost two digits of the constant reference specify which constant. The execution routines use the second word of the program header to locate the first constant and calculate the subscript in a floating point array equivalenced to KODE at which the referenced constant is stored. If the constant reference number is in the range 386 to 399, the constant is a system constant {1.3.2}. The execution routines calculate the subscript to access the referenced constant in floating point.

A variable reference is a 400 number. The rightmost two digits specify which link in the variable linkage area of the program (see Figure 2-1) corresponds to the variable. The first link is referenced by 401, the second by 402, etc. The execution routines use the first word of the program header to locate the first variable link. The link specified by the variable reference contains a number ranging from zero to 89. If the number is zero, the variable is currently undefined (no data has been assigned to the variable). If the number is positive, the variable is defined and the storage information for the variable is in the row of the data table {2.} specified by the number. The storage information consists of the number of floating point words comprising the data and the subscript of the first floating point number. The subscript is the number used when accessing the data using a floating point array equivalenced to the integer array KODE. Data is accessed using the floating point array only for arithmetic computation. Throughout the system floating point numbers are moved as two-word integers. The variable referencing mechanism is shown in Figure 4-11.

The system accumulator is referenced by a zero operand. The location and length of the system accumulator is contained in row 90 of the data table {2.}. The row contains zeros if storage is not currently allocated for the system accumulator.

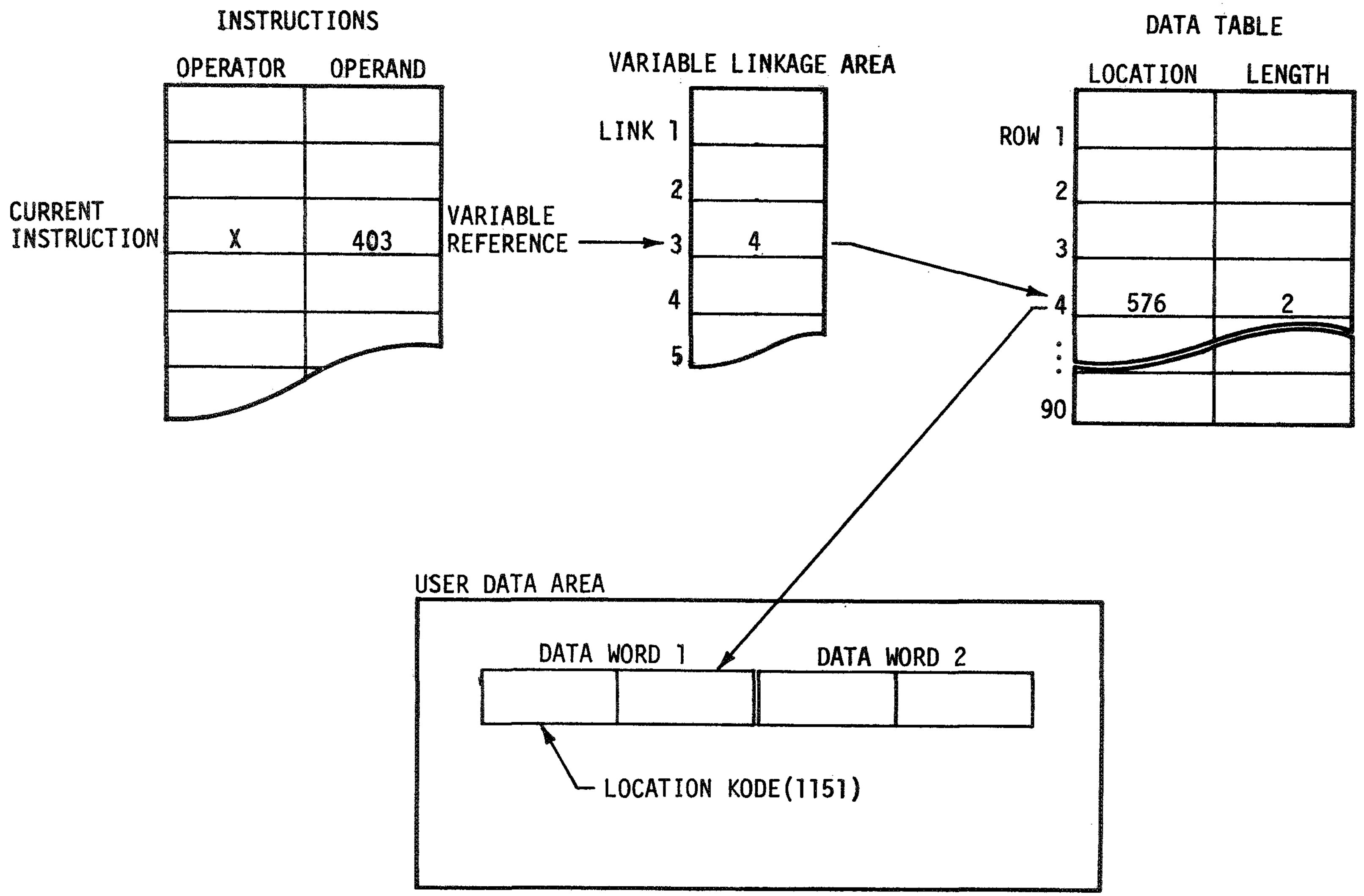


FIGURE 4-11. VARIABLE REFERENCING SYSTEM

4.6.2 Storage Allocation

Within this subsystem, the programs RTN, STV, LSG, and TRG request changes in storage allocation either to obtain new storage or to release storage for further use. Whenever storage allocation is performed, the following are updated by the programs in subsystem 4:

- Data table entry count {10.}
- Entries in the data table {2.} affected by the change.

(The storage allocation subprograms update the data storage count {12.}.) The updating of the data storage information by the programs in subsystem 4 will not be repeated in the individual program descriptions.

4.6.3 Program GETOP

The Assembler language program GETOP, shown in Figure 4-12, unpacks the interpreter instruction specified by the current location {4. }. The instruction is separated into the operator code {3.1-2} and the operand code {3.2-2} which are, respectively, in the leftmost seven bits and the remaining nine bits of the word. The operator class {5. } and subclass {3.7-2} are set. Where no subclass is specified for the operator, the subclass is set to zero. The operators, classes, and subclasses are listed in the description of subsystem 4 (see Section 3.4) and the operator codes are presented in Table 2-4.

4.6.4 Program RTN

The tasks of program RTN are to return control from a called user program to the calling program and to output system error messages. The execution of these tasks is controlled by the type of entry to RTN {3.7-3} which indicates the following:

1 - return from user program: execute interpreter instruction exit

2 - return from all called user programs to conversational mode program and output error message

3 - output error message.

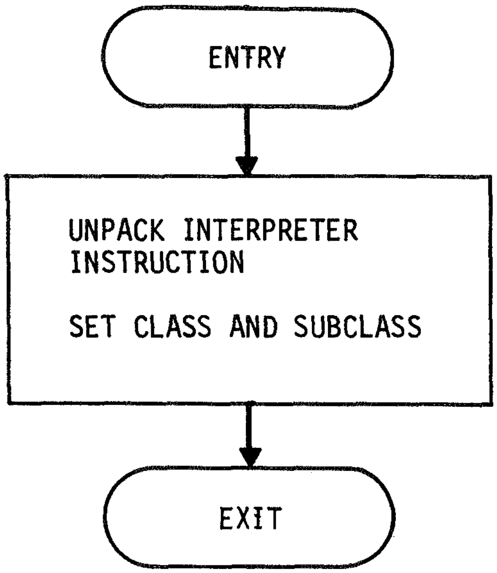


FIGURE 4-12. PROGRAM GETOP

The three entry types are described in the order in which they appear in Figure 4-13. For all entries, the program initially sets the output parameter to one. The possible values and meanings for the parameter are

1 - continue execution of interpreter instructions

2 - route system flow to subsystem 2 for reentry of source statement

3 - route system flow to subsystem 1 for normal program and data initialization.

4.6.4.1 Entry Type 3 - Program RTN reads that column of the error message control table {22.5} specified by the error indicator {9.} to obtain the record number at which the error message begins and the number of records on the system control file {22.} occupied by the error message. The message is then read and output on the typewriter. The error messages are listed in Appendix A. (Error message number 29 is output directly by program RTN.) If the error is number 54, indicating the maximum number of program statements have been entered, the statement entry mode {16.} is checked. If the system is in the execute mode, the previous variable count {1.1.3} is set to zero and the output parameter is set to three. If the system is in the SUPPRESS or EDIT mode, the previous mode program construction information (see Section 4.3.1.2) is restored from the system working file (either {20.4} or {20.5}).

If the card status {1.1.4} is two, indicating an error was detected during the translation of a program entered on cards, the statement number is output. The number is obtained from the row in the statement index table {1.2-1.3} indicated by the statement count {18.}. The conversational mode program construction information is then restored from the system working file {20.5}. For all errors, if the statement count is one, the output parameter is set to three; otherwise, the parameter is set to two. It should be noted that the error indicator {9.} is not cleared by program RTN.

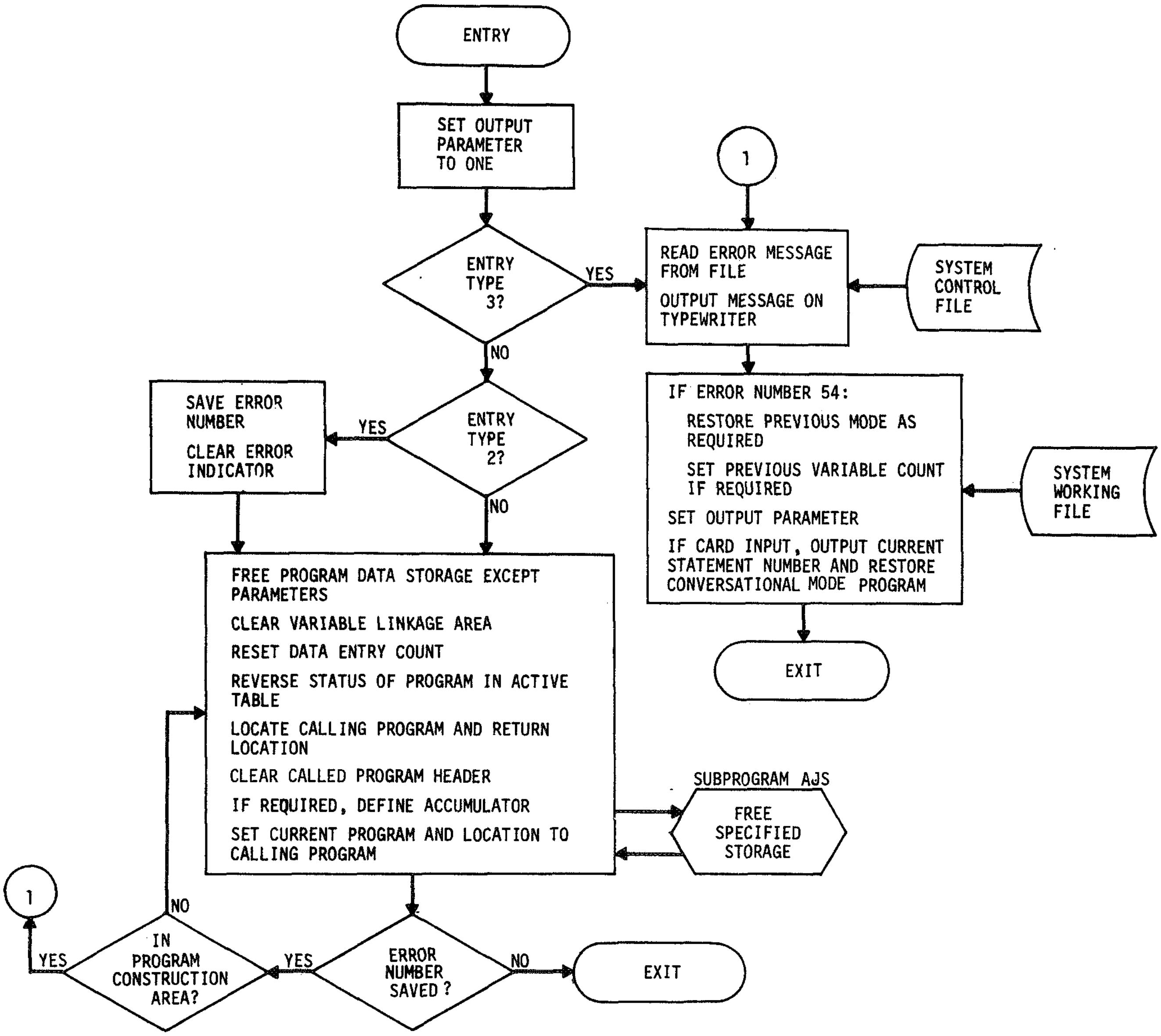

ENTRY
SET OUTPUT PARAMETER TO ONE
ENTRY TYPE 3?
YES
NO
1
READ ERROR MESSAGE FROM FILE
OUTPUT MESSAGE ON TYPEWRITER
SYSTEM CONTROL FILE
ENTRY TYPE 2?
YES
NO
SAVE ERROR NUMBER
CLEAR ERROR INDICATOR
IF ERROR NUMBER 54:
RESTORE PREVIOUS MODE AS REQUIRED
SET PREVIOUS VARIABLE COUNT IF REQUIRED
SET OUTPUT PARAMETER
IF CARD INPUT, OUTPUT CURRENT STATEMENT NUMBER AND RESTORE CONVERSATIONAL MODE PROGRAM
SYSTEM WORKING FILE
EXIT
FREE PROGRAM DATA STORAGE EXCEPT PARAMETERS
CLEAR VARIABLE LINKAGE AREA
RESET DATA ENTRY COUNT
REVERSE STATUS OF PROGRAM IN ACTIVE TABLE
LOCATE CALLING PROGRAM AND RETURN LOCATION
CLEAR CALLED PROGRAM HEADER
IF REQUIRED, DEFINE ACCUMULATOR
SET CURRENT PROGRAM AND LOCATION TO CALLING PROGRAM
SUBPROGRAM AJS
FREE SPECIFIED STORAGE
1
NO
YES
IN PROGRAM CONSTRUCTION AREA?
YES
ERROR NUMBER SAVED ?
NO
EXIT

FIGURE 4-13. PROGRAM RTN

4.6.4.2 Entry Type 2 - Program RTN saves the error number and clears the error indicator. The functions described for entry type 1 are performed until the system has returned execution through all the called user programs to the conversational mode program in the program construction area {1. 1} (until the current program pointer {8. } is one). The error indicator is then restored and all the actions described for entry type 3 are taken.

4.6.4.3 Entry Type 1 - When executing an exit interpreter instruction, program RTN releases all storage in the user data area {1. 3. 1} which is local to the user program: all storage for temporary variables and variables in the program which are not parameters is released. The data table entry count {10. } is adjusted accordingly. The program variable linkage area is set to all zeros so that the program can be reexecuted without reading a fresh copy from the program file. The program entry in the active table {1. 2-2. 1} is changed to indicate the program is no longer in the execution chain. This is done by making the entry in column 2 negative. The fourth word in the user program contains the program active number which is the row in the active table containing the entry for the program. (See Figure 2=1 for the program structure.) The fifth word of the header contains the active number for the calling program. Using this pointer to the active table, the current location of the calling program is found. The fourth word of the called program header contains the location relative to the calling program header at which execution is continued. The current program location {8. } is set to the location of the calling program and the current location {4-1} is set to the current program location plus the relative location for the return. The fourth and fifth words of the called program are set to zero. If the accumulator is not currently assigned, it is set to length one.

4.6.5 Program JMP

Program JMP executes the call to a user program. The functions of JMP, shown in Figure 4-14, can be grouped into the eight tasks described below. Unless otherwise stated, the tasks are executed in order.

4.6.5.1 Task 1 - If Overlay I {1.2-1} is in core, the overlay, except for the statement conversion area {1.2-1.1}, is saved on the system working file {20.6}. If the active program count {19.} is zero, Overlay II {1.2-2} is then initialized in core by setting KODE(451) to one, the active table {1.2-2.1} to all zeros, and the active area pointer {1.2-2.2} to 503. However, when the active program count is positive, Overlay II is restored from the system working file {20.7}. The overlay status {3.8} is set to two.

4.6.5.2 Task 2 - The first operand for a call instruction is a program reference number. It is 100 plus the column in the program call table which contains the name of the program to be called. Using the third word of the header (see Figure 2-1), the program name and the previous active number are located in the program call table of the calling program. If the previous active number is positive, indicating the program has been called previously, the program name is checked with the indicated entry in the active table {1.2-2.1}. If the names agree, the active number is current and task 8 is executed.

4.6.5.3 Task 3 - This task is performed only if there are programs in the active table. The active table is searched to find a match to the name of the program being requested. If the name appears in the active table, task 7 is performed next.

4.6.5.4 Task 4 - This task is performed only if the user program requested has not been found in the active table by tasks 2 or 3. The program control table is read from the user program file {21.4} and searched to locate the requested program. The record

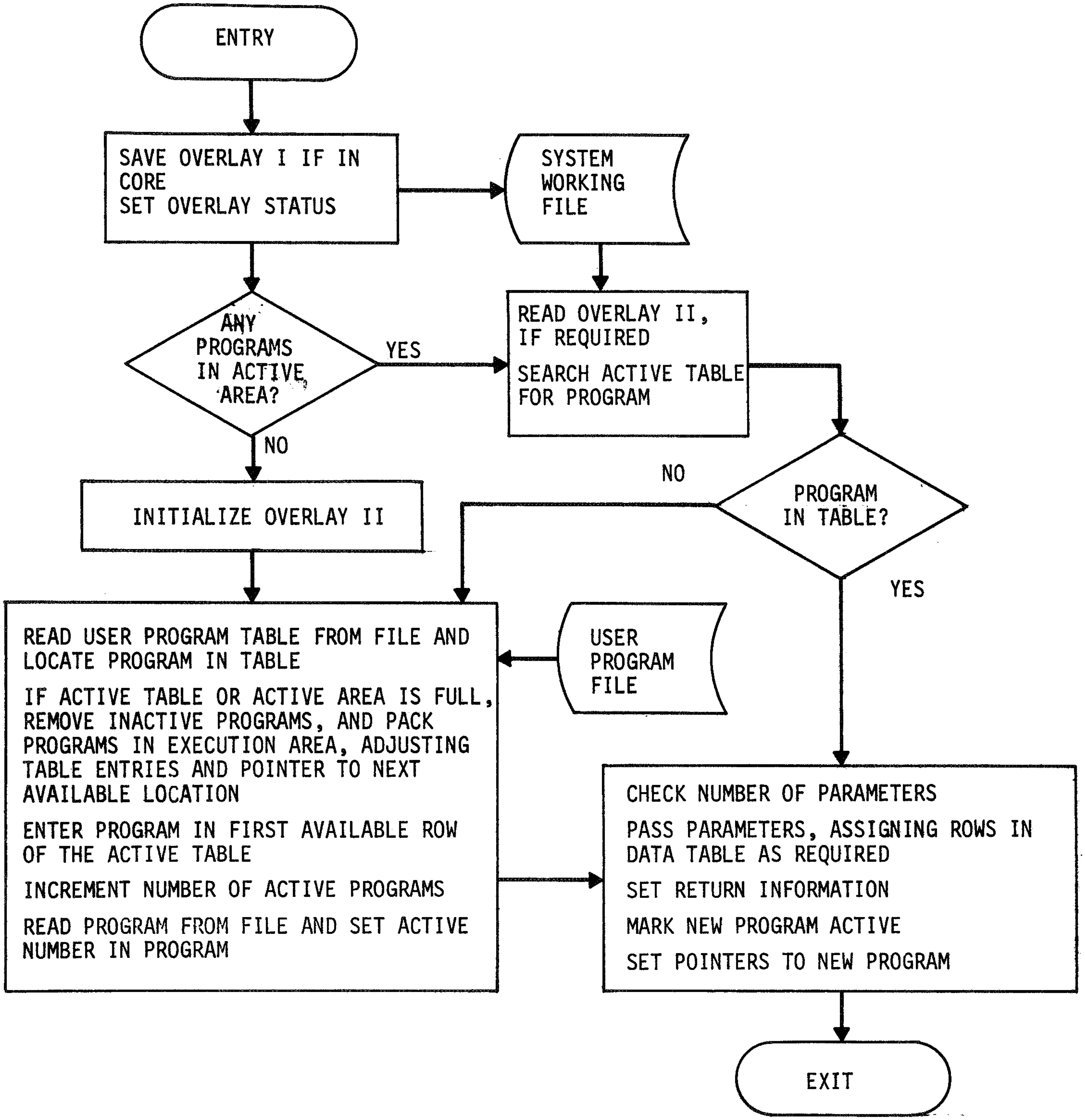

ENTRY
SAVE OVERLAY I IF IN CORE
SET OVERLAY STATUS
SYSTEM WORKING FILE
ANY PROGRAMS IN ACTIVE AREA?
YES
NO
READ OVERLAY II, IF REQUIRED
SEARCH ACTIVE TABLE FOR PROGRAM
INITIALIZE OVERLAY II
PROGRAM IN TABLE?
NO
YES
READ USER PROGRAM TABLE FROM FILE AND LOCATE PROGRAM IN TABLE
IF ACTIVE TABLE OR ACTIVE AREA IS FULL, REMOVE INACTIVE PROGRAMS, AND PACK PROGRAMS IN EXECUTION AREA, ADJUSTING TABLE ENTRIES AND POINTER TO NEXT AVAILABLE LOCATION
ENTER PROGRAM IN FIRST AVAILABLE ROW OF THE ACTIVE TABLE
INCREMENT NUMBER OF ACTIVE PROGRAMS
READ PROGRAM FROM FILE AND SET ACTIVE NUMBER IN PROGRAM
USER PROGRAM FILE
CHECK NUMBER OF PARAMETERS
PASS PARAMETERS, ASSIGNING ROWS IN DATA TABLE AS REQUIRED
SET RETURN INFORMATION
MARK NEW PROGRAM ACTIVE
SET POINTERS TO NEW PROGRAM
EXIT

FIGURE 4-14. PROGRAM JMP

location and the length of the internal form of the program are obtained, respectively, from the fourth and fifth columns of the table. If the active table is full (contains ten entries) or if the program will not fit in core, task 5 is executed. The program will not fit in core if its length plus the active area pointer {1.2-2.2} is greater than 1, 140 which is the beginning subscript of the data area {1.3}. If the program can be brought into core, task 6 is executed.

4.6.5.5 Task 5 - In this task, programs in the active area are packed to provide space for additional programs. The following actions are taken:

- Delete all inactive programs from the active table by setting the entry row to zero. A program is inactive (not in the current execution chain) if column 2 is negative.
- For each entry deleted, decrement the active program count {19.}
- Set the active area pointer {1.2-2.2} to 503
- Move all programs remaining in the execution area to the beginning of the area, removing unused words between them and updating the active area pointer and column 1 of the active table for each program moved.

4.6.5.6 Task 6 - The program located on the user program file by task 4 is assigned the active number corresponding to the first unused row in the active table {1.2-2.1}. The program name is copied from the program call table into the third through fifth columns of the assigned row, column 1 is set to the current value of the active area pointer {1.2-2.2}, and column 2 is set to minus the length of the program. The program is read from the user program file into the program execution area {1.2-2.3} at the place specified by the active area pointer. This pointer is then incremented to the first word beyond the program. The assigned active number is entered into the fifth word of the program header (see Figure 2-1).

4.6.5.7 Task 7 - The active number for the called program is entered into the program call table of the calling program (see Section 4.6.5.2).

4.6.5.8 Task 8 - This task passes parameters between the user programs. If parameters were entered in the user program call statement, they appear as variable references (one per word) in the words following the call instruction. The number of parameters appearing is validated by the number specified in the fourth word of the called program header. For each variable in the parameter list which has a zero value in the linkage area, the next available row in the data table {2.} is assigned for future storage, the link is filled with the row number, and the data table entry count {10.} is incremented. Parameters are passed in order and to the first variables (those with lowest variable reference numbers beginning with 401) in the called program. For each variable in the parameter list, the corresponding link from the variable linkage area in the calling program is set in the next variable link in the called program. After the parameter links have been copied, the information required to return to the calling program is set in the following words of the called program header:

Word 6 - active number for calling program, obtained from word 5 of the calling program header except for the conversational mode program which has active number zero

Word 7 - set to the location relative to the program location (the first word of the calling program header) of the last parameter in the calling sequence.

The entry for the called program is marked active, the current program pointer {8.} is set to point to the header of the called program, and the current location {4-1} is set to the value of the current program pointer plus six.

4.6.6 Program STV

Program STV, flowcharted in Figure 4-15, executes the interpreter instruction INPUT (see Table 2-4). The first task performed by STV is to request entry of the data. The name of the variable to be defined is in the three words following the INPUT instruction and is packed two AMTRAN coded characters per word. The program STV reads the EBCDIC table from the system control file {22.4}, converts the variable name to EBCDIC, and outputs the variable name of the typewriter preceded by the word ENTER. The data storage currently assigned to the variable is located through the variable link to the data table {2.} (see Section 4.6.1). If a link does not exist, the next available row in the data table is assigned to the variable, the data table entry count {10.} is incremented, and the link is filled in.

Program STV uses a modified form of the AMTRAN character codes. Table 4-6 contains the codes used by STV. The input to STV can be on cards or from the console typewriter. Sense switch 15 controls the device selection:

Switch 15 OFF (down) - typewriter input
Switch 15 ON (up) - card input.

If the input is on cards, program STV reads an array from the system working file {22.3} containing the EBCDIC codes for the characters STV recognizes. Card input is processed one card at a time. The characters are converted from EBCDIC to the codes in Table 4-6 and the code string is scanned.

If the input is from the typewriter, the entire statement is read by the subprogram KYBRD. The output from this routine is in AMTRAN character codes which STV then converts to the codes in Table 4-6 before scanning the entire statement.

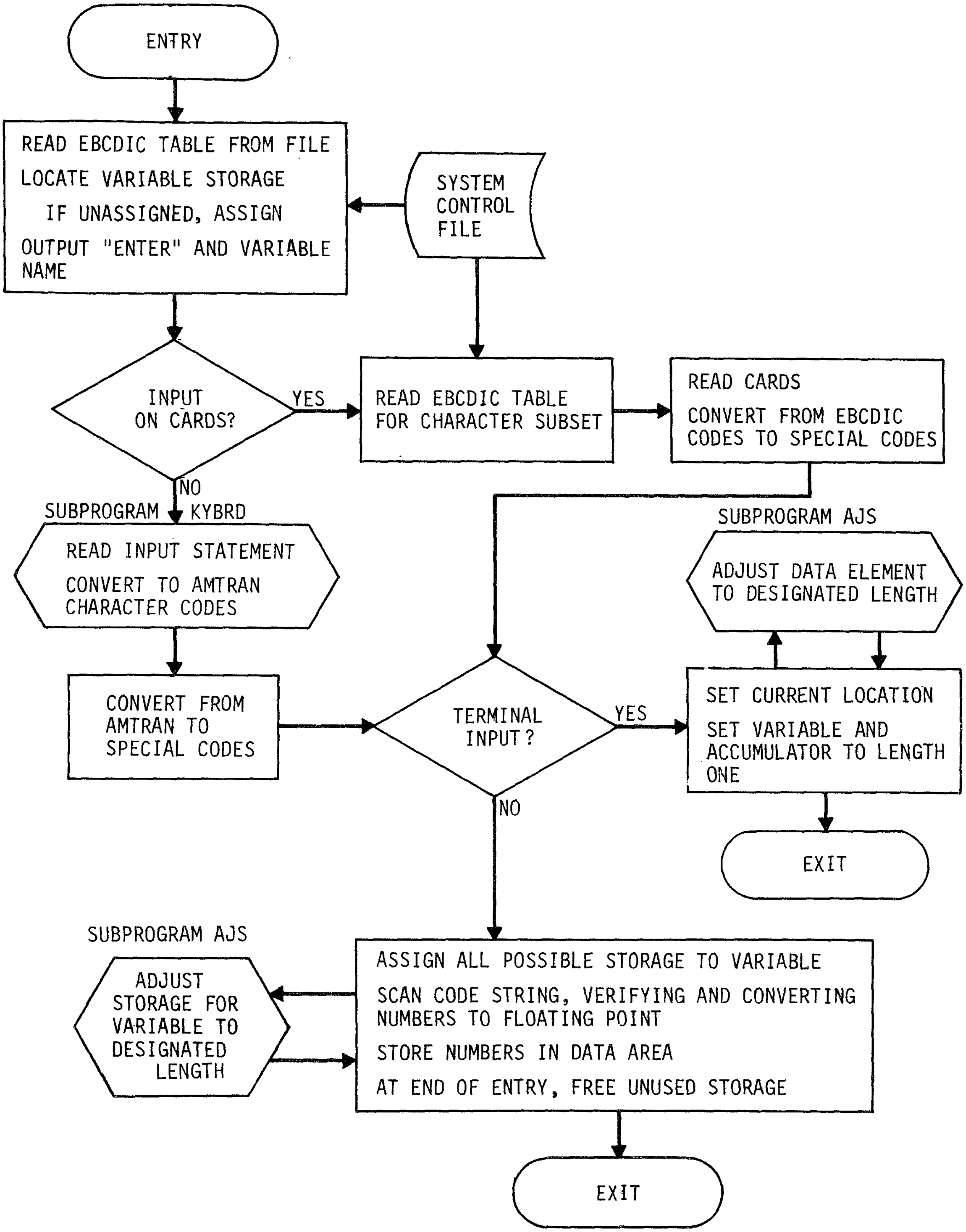

ENTRY
READ EBCDIC TABLE FROM FILE
LOCATE VARIABLE STORAGE
IF UNASSIGNED, ASSIGN
OUTPUT "ENTER" AND VARIABLE NAME
SYSTEM CONTROL FILE
INPUT ON CARDS?
YES
NO
READ EBCDIC TABLE FOR CHARACTER SUBSET
READ CARDS
CONVERT FROM EBCDIC CODES TO SPECIAL CODES
SUBPROGRAM KYBRD
READ INPUT STATEMENT
CONVERT TO AMTRAN CHARACTER CODES
SUBPROGRAM AJS
ADJUST DATA ELEMENT TO DESIGNATED LENGTH
CONVERT FROM AMTRAN TO SPECIAL CODES
TERMINAL INPUT?
YES
NO
SET CURRENT LOCATION
SET VARIABLE AND ACCUMULATOR TO LENGTH ONE
EXIT
SUBPROGRAM AJS
ADJUST STORAGE FOR VARIABLE TO DESIGNATED LENGTH
ASSIGN ALL POSSIBLE STORAGE TO VARIABLE
SCAN CODE STRING, VERIFYING AND CONVERTING NUMBERS TO FLOATING POINT
STORE NUMBERS IN DATA AREA
AT END OF ENTRY, FREE UNUSED STORAGE
EXIT

FIGURE 4-15. PROGRAM STV

TABLE 4-6. CHARACTERS AND CORRESPONDING CODES USED BY PROGRAM STV

Character	Code
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
Blank	10
-	11
.	12
,	13
E	14
/	15

The input to STV is a sequence of numeric constants separated by commas or blanks. STV also accepts floating point entries in which the exponent is in one of the forms

E d
E dd
Edd
Ed
E-d
E-dd

where d represents a digit. The task of STV is to scan the string of character codes and convert the numbers to floating point.

When the first number has been converted to floating point, all available storage, as indicated by the current length of the variable and the data storage count {12.}, is allocated to the variable. The converted number is stored as the first element of the variable. As each subsequent number is converted to floating point, it is stored in the area allocated for the variable as the next data element.

If an error is detected while STV is processing a card input, the program deletes those numbers already stored which appeared on the current card and requests reentry of the card. If an error is detected when reading from the typewriter, the last number accepted by STV and a request for reentry of the remaining numbers are output.

The termination of input is indicated by an eof on the typewriter and by a double slash (//) on cards. When the termination characters are recognized, any storage not used for the variable is returned to available storage and the entry in the data table {2.} for the variable is set to the correct length.

When numbers are not input prior to the termination characters, the user is terminating an input loop. If storage has been allocated for the variable, it is reduced to length one. The accumulator is set to length one. If the pointer to the last interpreter branch instruction {17.} is set, the current location {4-1} is set to its value; otherwise, the current location is not changed.

4.6.7 Program WRT

Program WRT, shown in Figure 4-16, executes the class 4 interpreter instructions: TYPEOUT, TYPE, and PUNCH (see Table 2-4 for the operator codes). Program WRT is provided the operator code {3.1-2}, the code for the first operand {3.2-2}, and the current location {4-1} of the interpreter instruction to be executed.

For each of the class 4 operators, the user selects the output device using sense switch 0:

Switch 0 OFF (down) - typewriter output
Switch 0 ON (up) - printer output.

For TYPE and PUNCH, the user selects the output format using sense switch 1:

Switch 1 OFF (down) - fixed point format
Switch 1 ON (up) - floating point format.

The operand {3.2-2} for TYPEOUT is the number of words immediately following the TYPEOUT instruction which contain the message to be output. The message consists of characters in AMTRAN character codes which are packed two characters per word (each character occupies eight bits). Program WRT does the following tasks to execute the TYPEOUT operator:

- Read EBCDIC table from the system control file {22.4}
- For each word specified by the word count (the operand {3.2-2})
  - ▲ Unpack characters
  - ▲ Place equivalent EBCDIC codes in an output array
  - ▲ If a carriage return is encountered, output the array
  - ▲ On the last character, output the array
- Set the current location {4-1} to the last word containing characters in the message: set to the current location upon entry to WRT plus the operand {3.2-2}.

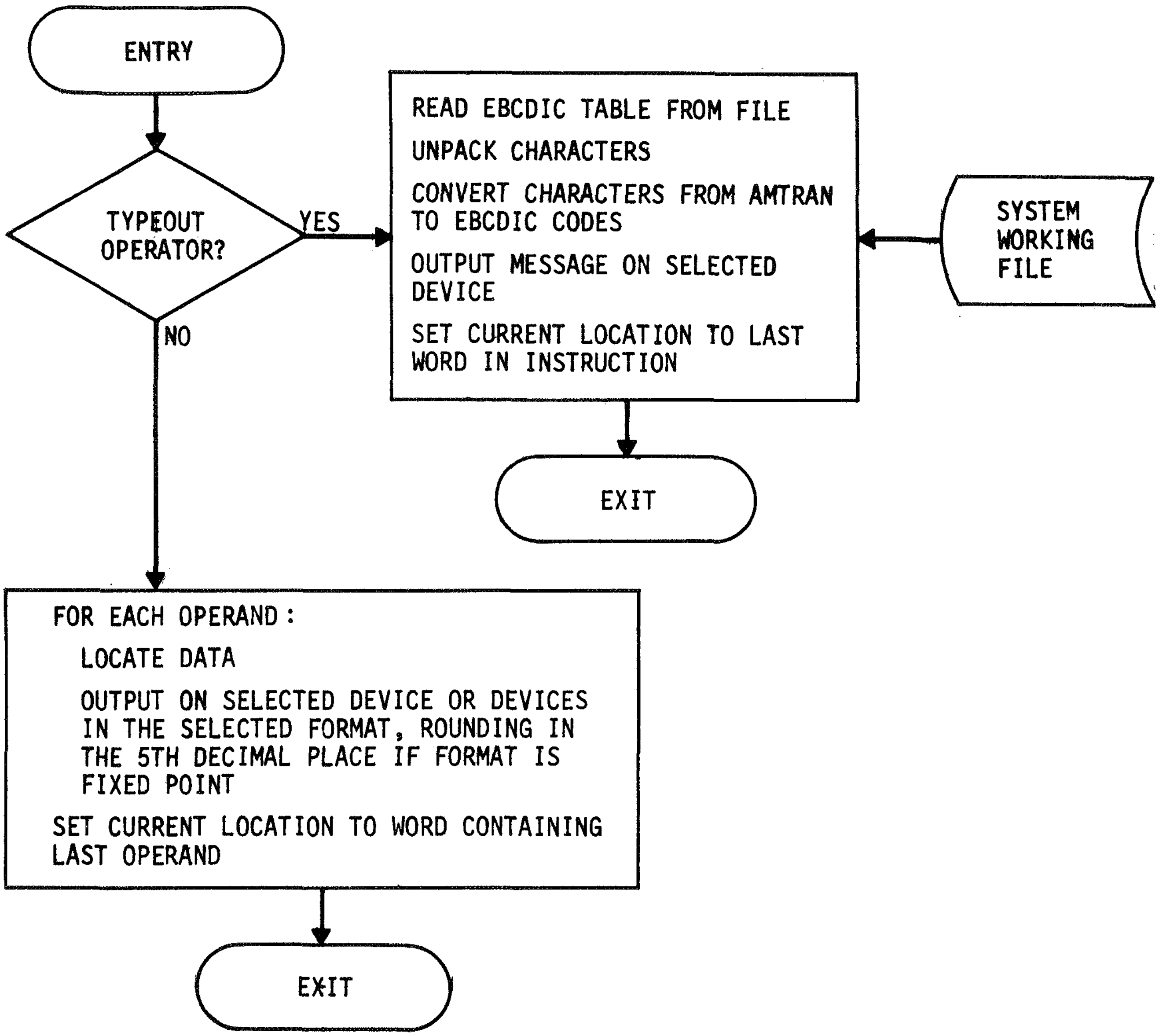

ENTRY
TYPEOUT OPERATOR?
YES
NO
READ EBCDIC TABLE FROM FILE
UNPACK CHARACTERS
CONVERT CHARACTERS FROM AMTRAN TO EBCDIC CODES
OUTPUT MESSAGE ON SELECTED DEVICE
SET CURRENT LOCATION TO LAST WORD IN INSTRUCTION
SYSTEM WORKING FILE
EXIT
FOR EACH OPERAND:
LOCATE DATA
OUTPUT ON SELECTED DEVICE OR DEVICES IN THE SELECTED FORMAT, ROUNDING IN THE 5TH DECIMAL PLACE IF FORMAT IS FIXED POINT
SET CURRENT LOCATION TO WORD CONTAINING LAST OPERAND
EXIT

FIGURE 4-16. PROGRAM WRT

The TYPE and PUNCH instructions can have multiple operands which can be either constant or variable references. When only one operand is specified, it may refer to the system accumulator. For each operand, the associated data is located (see Section 4.6.1). The data is then output on the selected device in the selected format. If the operator is PUNCH, the data is output on cards as well as on the printer or typewriter. If the format is floating point, the numbers are first rounded by adding 0.00005 to each. The fixed point output contains four significant digits to the right of the decimal point and up to seven digits preceding the decimal point. For TYPE, eight numbers are output per line; for PUNCH, six numbers are punched per card and output per line. Six significant figures are output in the floating point format with seven numbers per line or card. Before exiting from program WRT, the current location {4-1} is set to the word containing the last operand for a TYPE or PUNCH operation.

The user can suppress or terminate the execution of any of the three operations by sense switch 13:

Switch 13 OFF (down) - normal execution
Switch 13 ON (up) - terminate execution.

Program WRT checks the status of sense switch 13 and terminates execution if required. The current location {4-1} is always set as if normal execution had been completed.

4.6.8 Program LSG

Program LSG executes the class 5 interpreter instructions which are grouped into the following subclasses:

1 - load accumulator, load accumulator and free temporary
2 - SIN, COS, EXP, SQRT, negation
3 - +, *, /, -, &
4 - GO TO, LT, GT, EQ, NE, GE, LE
5 - store accumulator
6 - free temporary.

The program is provided the subclass {3.7-2}, operator {3.1-2}, first operand {3.2-2}, and current location {4-1}. LSG sets an output parameter to one of the following:

1 - continue normal execution

2 - route system flow to subsystem 1 for normal program and data initialization

3 - continue normal execution without clearing the branch pointer {17.} before executing the next interpreter instruction

4 - an error has been detected, output error message.

Before any operators are executed, the output parameter is initialized to one. For each operator, the operand is validated using the operand requirements listed in Table 2-4. For all operators except the store, data must be currently assigned to the operand. The functions of LSG, shown in Figure 4-17, are described by subclass.

4.6.8.1 Subclass 1 - For both of the operators in this subclass, the storage information for the operand is located in the data table {2.} (see Section 4.6.1). For the load accumulator and free temporary instruction, the accumulator is adjusted to length zero and the entries in the data table {2.} for the accumulator and the temporary variable are exchanged: the contents of the respective rows are exchanged.

The load accumulator instruction may have one or two additional operands which appear in the one word or two words immediately following the load instruction. If there are multiple operands, the actual value(s) of the operand(s) are obtained and the current location {4-1} is incremented to the word containing the last operand. Each additional operand specifies a subscript in the variable to be loaded (the first operand). If two subscripts appear, they specify a range of subscripts. The subscript or subscripts are checked to make sure they are in the

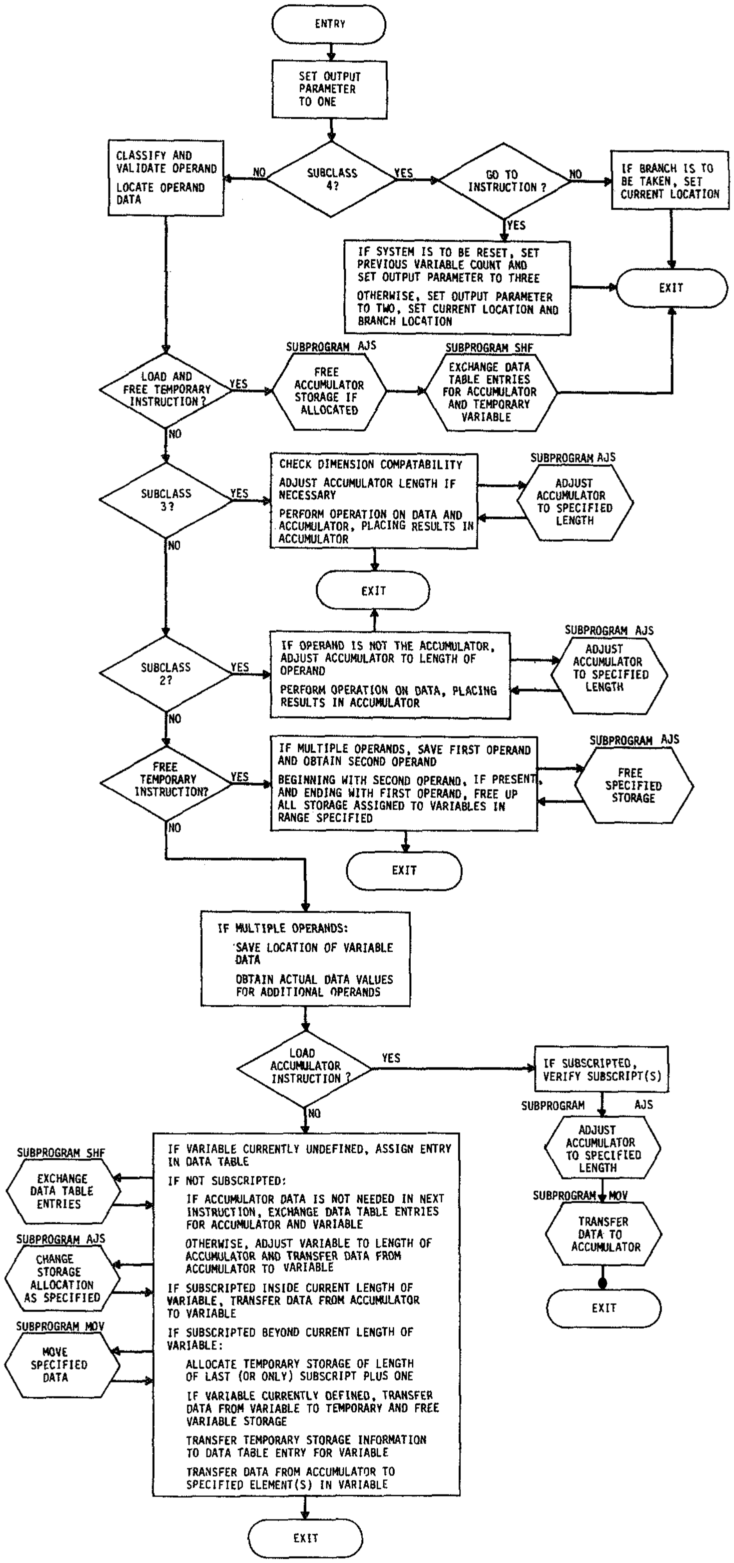

ENTRY
SET OUTPUT PARAMETER TO ONE
SUBCLASS 4?
NO
YES
CLASSIFY AND VALIDATE OPERAND
LOCATE OPERAND DATA
GO TO INSTRUCTION ?
NO
IF BRANCH IS TO BE TAKEN, SET CURRENT LOCATION
YES
IF SYSTEM IS TO BE RESET, SET PREVIOUS VARIABLE COUNT AND SET OUTPUT PARAMETER TO THREE
OTHERWISE, SET OUTPUT PARAMETER TO TWO, SET CURRENT LOCATION AND BRANCH LOCATION
EXIT
LOAD AND FREE TEMPORARY INSTRUCTION ?
YES
SUBPROGRAM AJS
FREE ACCUMULATOR STORAGE IF ALLOCATED
SUBPROGRAM SHF
EXCHANGE DATA TABLE ENTRIES FOR ACCUMULATOR AND TEMPORARY VARIABLE
NO
SUBCLASS 3?
YES
CHECK DIMENSION COMPATABILITY
ADJUST ACCUMULATOR LENGTH IF NECESSARY
PERFORM OPERATION ON DATA AND ACCUMULATOR, PLACING RESULTS IN ACCUMULATOR
SUBPROGRAM AJS
ADJUST ACCUMULATOR TO SPECIFIED LENGTH
NO
EXIT
SUBCLASS 2?
YES
IF OPERAND IS NOT THE ACCUMULATOR, ADJUST ACCUMULATOR TO LENGTH OF OPERAND
PERFORM OPERATION ON DATA, PLACING RESULTS IN ACCUMULATOR
SUBPROGRAM AJS
ADJUST ACCUMULATOR TO SPECIFIED LENGTH
NO
FREE TEMPORARY INSTRUCTION?
YES
IF MULTIPLE OPERANDS, SAVE FIRST OPERAND AND OBTAIN SECOND OPERAND
BEGINNING WITH SECOND OPERAND, IF PRESENT, AND ENDING WITH FIRST OPERAND, FREE UP ALL STORAGE ASSIGNED TO VARIABLES IN RANGE SPECIFIED
SUBPROGRAM AJS
FREE SPECIFIED STORAGE
NO
EXIT
IF MULTIPLE OPERANDS:
SAVE LOCATION OF VARIABLE DATA
OBTAIN ACTUAL DATA VALUES FOR ADDITIONAL OPERANDS
LOAD ACCUMULATOR INSTRUCTION ?
YES
IF SUBSCRIPTED, VERIFY SUBSCRIPT(S)
SUBPROGRAM AJS
ADJUST ACCUMULATOR TO SPECIFIED LENGTH
SUBPROGRAM MOV
TRANSFER DATA TO ACCUMULATOR
EXIT
NO
SUBPROGRAM SHF
EXCHANGE DATA TABLE ENTRIES
SUBPROGRAM AJS
CHANGE STORAGE ALLOCATION AS SPECIFIED
SUBPROGRAM MOV
MOVE SPECIFIED DATA
IF VARIABLE CURRENTLY UNDEFINED, ASSIGN ENTRY IN DATA TABLE
IF NOT SUBSCRIPTED:
IF ACCUMULATOR DATA IS NOT NEEDED IN NEXT INSTRUCTION, EXCHANGE DATA TABLE ENTRIES FOR ACCUMULATOR AND VARIABLE
OTHERWISE, ADJUST VARIABLE TO LENGTH OF ACCUMULATOR AND TRANSFER DATA FROM ACCUMULATOR TO VARIABLE
IF SUBSCRIPTED INSIDE CURRENT LENGTH OF VARIABLE, TRANSFER DATA FROM ACCUMULATOR TO VARIABLE
IF SUBSCRIPTED BEYOND CURRENT LENGTH OF VARIABLE:
ALLOCATE TEMPORARY STORAGE OF LENGTH OF LAST (OR ONLY) SUBSCRIPT PLUS ONE
IF VARIABLE CURRENTLY DEFINED, TRANSFER DATA FROM VARIABLE TO TEMPORARY AND FREE VARIABLE STORAGE
TRANSFER TEMPORARY STORAGE INFORMATION TO DATA TABLE ENTRY FOR VARIABLE
TRANSFER DATA FROM ACCUMULATOR TO SPECIFIED ELEMENT(S) IN VARIABLE
EXIT

FIGURE 4-17. PROGRAM LSG

current subscript range of the variable. Program LSG then reallocates the system accumulator (row 90 of the data table {2.}) to the length of the data to be loaded and transfers the specified data from the variable storage to the system accumulator.

4.6.8.2 Subclass 2 - The operators in this class may have the system accumulator specified as the operand. If the operand is not the accumulator, the storage information for the operand is obtained (see Section 4.6.1) and the system accumulator is adjusted to the length of the operand data. The individual interpreter operation specified is then performed in floating point on the data or accumulator and the result is placed in the system accumulator.

4.6.8.3 Subclass 3 - For the binary operations, program LSG locates the storage information for the operand. The length of the operand and of the system accumulator are compared. If the lengths are the same or if the lengths are not the same but either one has length one, the operation is valid. If the operand and accumulator are the same length, the specified operation is performed between corresponding elements. (For a binary operation, the accumulator is the first operand and the operand appearing with the operator is the second operand.) If the lengths are unequal and the operand has length one, the operation is performed between the scalar operand and each element in the accumulator. If the lengths are unequal and the accumulator is of length one, each element in the operand data is operated with the constant after the accumulator has been expanded to the length of the operand. For all operators in subclass 3, the result is placed in the system accumulator.

4.6.8.4 Subclass 4 - For the relational operators in this subclass, a branch to a nonsequential interpreter instruction may be performed, based on the value of the single element in the accumulator. The following list identifies each relation and the accumulator condition upon which a branch is taken:

LT - positive or zero
GT - negative or zero
EQ - positive or negative
NE - zero
GE - negative
LE - positive.

When a branch is taken, the current location {4-1} is incremented by the operand minus one.

On the GO TO instruction, the user may terminate execution and cause a reset of the system using sense switch 14:

Switch 14 OFF (down) - continue execution
Switch 14 ON (up) - reset system.

If a reset is required, program LSG sets the previous variable count {1.1.3} to zero and routes system flow to subsystem 1, entry 1 by setting the output parameter to three.

If a reset is not required, program LSG does the following:

- Set the pointer to the last executed branch instruction {17.} to the current location {4-1}
- Set the output parameter to two
- Increment the current location by the operand minus one if the operand is not zero.

4.6.8.5 Subclass 5 - If a link to the data table {2.} does not exist for the variable operand (see Section 4.6.1), the next available row in the data table as specified by the data table entry count {10.} is assigned to the variable, the entry count is incremented, and the link is set for the variable in the variable linkage area of the user program being executed.

The store instruction may have one or two additional operands, each specifying a subscript. Program LSG obtains the value of any additional operands. If the variable is not subscripted, program LSG will exchange the data table entries for the variable and the system accumulator if one of the following conditions exists:

- Storage is not currently allocated for the variable
- The store instruction is immediately followed by an instruction in one of the following classes (the classes and subclasses are described in the description of the subsystem, Section 3.4):

  ▲ Class 2
  ▲ Class 3
  ▲ Class 5, subclass 1
  ▲ Class 6, subclasses 1 and 4
  ▲ Class 7

- The store instruction is immediately followed by an instruction of one of the following types with an operand other than the system accumulator

  ▲ Class 4
  ▲ Class 5, subclass 2
  ▲ Class 6, subclasses 2 and 3
  ▲ Class 8

- The program being executed is in the program construction area {1.1} (the current program pointer {8.} is one) and the current instruction, as specified by the current location {4-1}, is the last instruction to be executed {6.}.

If the data table entries cannot be exchanged, the amount of storage allocated for the variable is adjusted to equal the length of the system accumulator and the data in the accumulator is duplicated in the variable storage.

If the variable is subscripted and the subscripting refers to an element or elements currently existing in the variable data, the data in the system accumulator is transferred to the data element or elements of the variable specified by the subscripting.

If the variable is subscripted and the subscripting is beyond the current length of the variable, temporary storage of length equal to the higher (or only) subscript is allocated. If storage is currently allocated for the variable, the data in that storage is transferred to the temporary storage and the storage assigned to the variable is freed. The storage information (location and length) for the temporary is transferred to the variable entry in the data table. The data in the accumulator is transferred to the specified element(s) of the variable.

4.6.8.6 Subclass 6 - The free temporary instruction may have either one or two operands. If one operand appears, any data storage assigned to the operand is made available for further use. If two operands appear, any storage allocated to the two operands and to variables with reference numbers in the range delimited by the two operands is returned to free storage.

4.6.9 Program TRG

Program TRG executes the class 6 interpreter instructions. The program is provided the operator code {3.1-2}, the first operand {3.2-2}, the current location {4-1}, and the subclass {3.7-2} which indicates the following instructions:

1 - ARRAY (RANGE)
2 - MIN, MAX, INTERVALS, SUMF
3 - LN, ATAN, ABS, TANH, SUM, MAGNITUDE
4 - SHIFT
5 - exponentiation.

For all operators, program TRG validates the operand according to Table 2-4. The location and length of the operand data is obtained using the method described in Section 4.6.1. Program TRG then performs the separate functions shown in Figure 4-18 for the various operators.

4.6.9.1 ARRAY - For the ARRAY function, program TRG obtains the values of the three scalar operands. The system accumulator

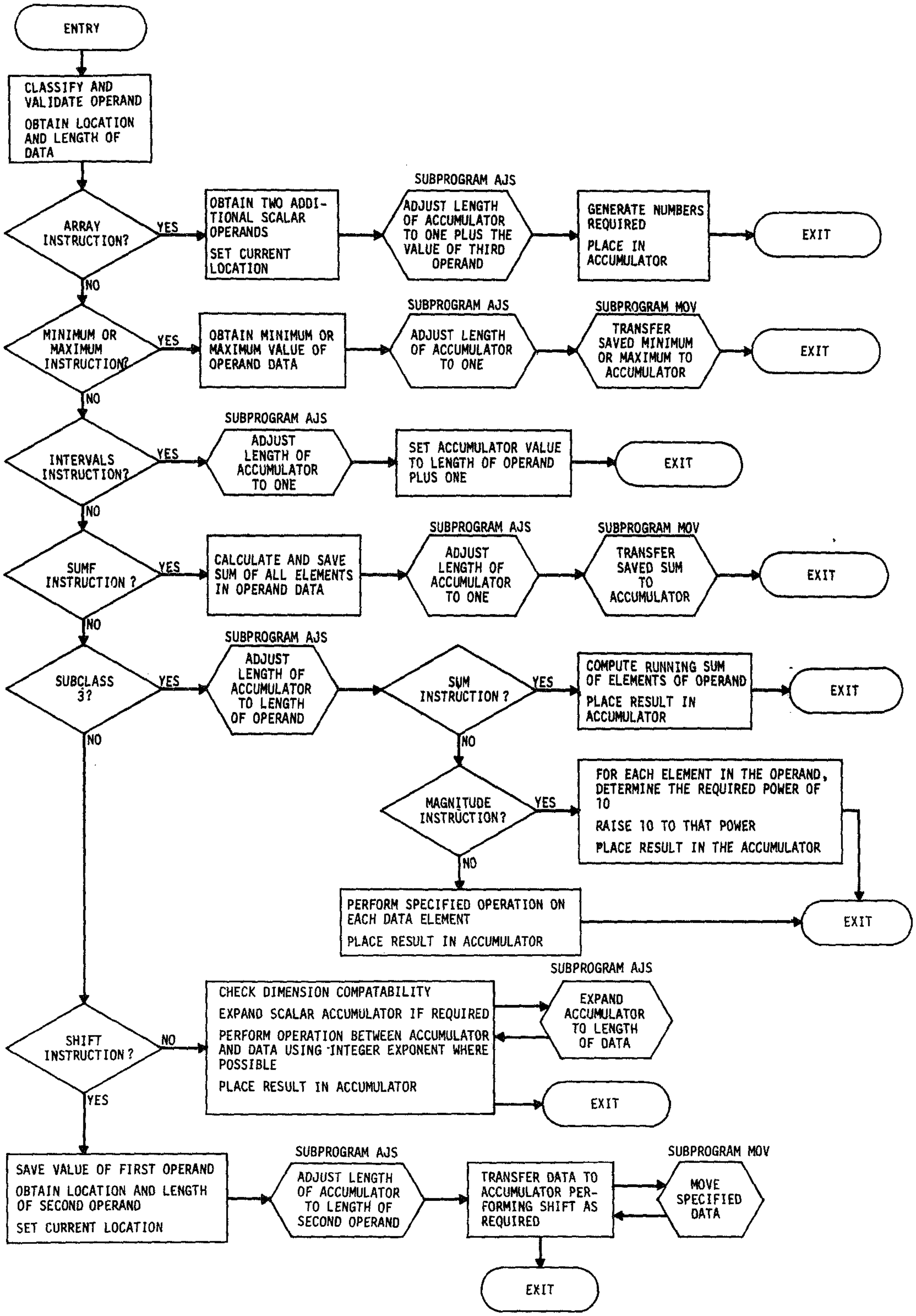

ENTRY
CLASSIFY AND VALIDATE OPERAND
OBTAIN LOCATION AND LENGTH OF DATA
ARRAY INSTRUCTION?
YES
NO
OBTAIN TWO ADDITIONAL SCALAR OPERANDS
SET CURRENT LOCATION
SUBPROGRAM AJS
ADJUST LENGTH OF ACCUMULATOR TO ONE PLUS THE VALUE OF THIRD OPERAND
GENERATE NUMBERS REQUIRED
PLACE IN ACCUMULATOR
EXIT
MINIMUM OR MAXIMUM INSTRUCTION?
YES
NO
OBTAIN MINIMUM OR MAXIMUM VALUE OF OPERAND DATA
SUBPROGRAM AJS
ADJUST LENGTH OF ACCUMULATOR TO ONE
SUBPROGRAM MOV
TRANSFER SAVED MINIMUM OR MAXIMUM TO ACCUMULATOR
EXIT
INTERVALS INSTRUCTION?
YES
NO
SUBPROGRAM AJS
ADJUST LENGTH OF ACCUMULATOR TO ONE
SET ACCUMULATOR VALUE TO LENGTH OF OPERAND PLUS ONE
EXIT
SUMF INSTRUCTION ?
YES
NO
CALCULATE AND SAVE SUM OF ALL ELEMENTS IN OPERAND DATA
SUBPROGRAM AJS
ADJUST LENGTH OF ACCUMULATOR TO ONE
SUBPROGRAM MOV
TRANSFER SAVED SUM TO ACCUMULATOR
EXIT
SUBCLASS 3?
YES
NO
SUBPROGRAM AJS
ADJUST LENGTH OF ACCUMULATOR TO LENGTH OF OPERAND
SUM INSTRUCTION ?
YES
NO
COMPUTE RUNNING SUM OF ELEMENTS OF OPERAND
PLACE RESULT IN ACCUMULATOR
EXIT
MAGNITUDE INSTRUCTION?
YES
NO
FOR EACH ELEMENT IN THE OPERAND, DETERMINE THE REQUIRED POWER OF 10
RAISE 10 TO THAT POWER
PLACE RESULT IN THE ACCUMULATOR
PERFORM SPECIFIED OPERATION ON EACH DATA ELEMENT
PLACE RESULT IN ACCUMULATOR
EXIT
SHIFT INSTRUCTION ?
NO
YES
CHECK DIMENSION COMPATABILITY
EXPAND SCALAR ACCUMULATOR IF REQUIRED
PERFORM OPERATION BETWEEN ACCUMULATOR AND DATA USING INTEGER EXPONENT WHERE POSSIBLE
PLACE RESULT IN ACCUMULATOR
SUBPROGRAM AJS
EXPAND ACCUMULATOR TO LENGTH OF DATA
EXIT
SAVE VALUE OF FIRST OPERAND
OBTAIN LOCATION AND LENGTH OF SECOND OPERAND
SET CURRENT LOCATION
SUBPROGRAM AJS
ADJUST LENGTH OF ACCUMULATOR TO LENGTH OF SECOND OPERAND
TRANSFER DATA TO ACCUMULATOR PERFORMING SHIFT AS REQUIRED
SUBPROGRAM MOV
MOVE SPECIFIED DATA
EXIT

FIGURE 4-18. PROGRAM TRG

is set to a length equal to the third operand plus one. The floating point numbers are generated and stored in the system accumulator: the first element is set to the value of the first operand; the last element is set to the value of the second operand; and the remaining numbers are generated at equal intervals between, as determined by the third operand. The current location {4.1} is updated to the instruction word containing the third operand.

4.6.9.2 MIN or MAX - Program TRG obtains the minimum or maximum value in the operand data. The selected value is stored in the temporary register, KODE (1143) and KODE (1144), in the user data area {1.3.1}. The accumulator is adjusted to length one and the saved minimum or maximum is transferred to the accumulator.

4.6.9.3 INTERVALS - The system accumulator is adjusted to length one and set to the length of the operand data minus one.

4.6.9.4 SUMF - The elements in the operand data are summed and the total is stored in the data temporary, KODE (1143) and KODE (1144). The system accumulator is adjusted to length one and set to the total.

4.6.9.5 LN, ATAN, ABS, and TANH - The system accumulator is adjusted to the length of the operand. The specified operation is applied to each element of the operand and the result is placed in the system accumulator.

4.6.9.6 SUM - The system accumulator is adjusted to the length of the operand. A running summation is performed on the operand data, placing each sum in the corresponding position of the system accumulator.

4.6.9.7 MAGNITUDE - The system accumulator is adjusted to the length of the operand. For each element x in the operand, the following is computed:

- $k = \log_{10} x$
- if $k < 0$, $r = 10 ** (k-1)$
  if $k \geq 0$, $r = 10 ** k$

where r is the result placed in the system accumulator.

4.6.9.8 SHIFT - The SHIFT operator has two operands. Program TRG saves the value of the first operand and obtains the location and length of the second operand, which is the variable to be shifted. The current location {4-1} is incremented by one. The system accumulator is adjusted to the length of the second operand. The minimum shift is determined using the first operand and the length of the second operand. If the shift is zero or equal to the variable length, no shift is performed. The variable data is transferred directly to the accumulator. When a shift is required, the data is transferred to the accumulator, making the shift part of the transfer.

4.6.9.9 Exponentiation - The length of the system accumulator and the operand data are compared. If equal, each element in the system accumulator is raised to the power which is the corresponding element in the operand data. If the operand is a scalar, each number in the accumulator is raised to that power. If the system accumulator is a scalar and the operand is an array, the accumulator is expanded to the length of the array, the scalar is duplicated in each position of the accumulator, and the operation is performed. When performing the operation exponentiation, the number is raised to an integer power rather than a floating point power whenever possible. The result is placed in the system accumulator.

4.6.10 Program TAB

The interpreter instruction TAB is executed by this program which is shown in Figure 4-19. The program is provided the first operand {3.1-2} and the current location {4-1}. Program TAB outputs

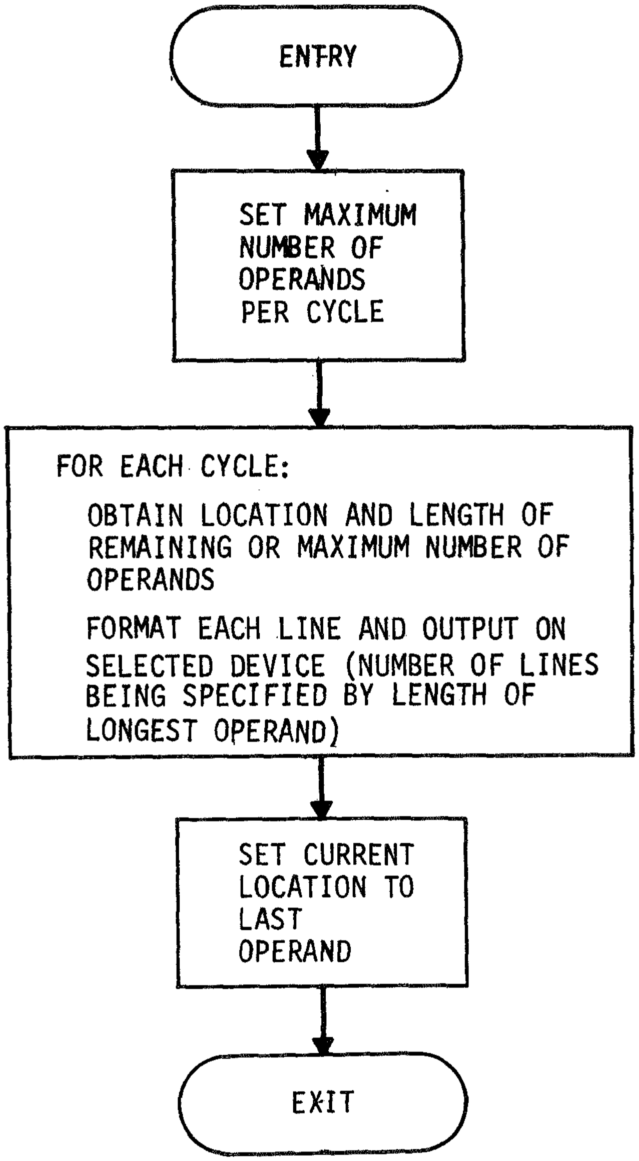

ENTRY
SET MAXIMUM NUMBER OF OPERANDS PER CYCLE
FOR EACH CYCLE:
OBTAIN LOCATION AND LENGTH OF REMAINING OR MAXIMUM NUMBER OF OPERANDS
FORMAT EACH LINE AND OUTPUT ON SELECTED DEVICE (NUMBER OF LINES BEING SPECIFIED BY LENGTH OF LONGEST OPERAND)
SET CURRENT LOCATION TO LAST OPERAND
EXIT

FIGURE 4-19. PROGRAM TAB

data in fixed point or floating point format, depending on the status of sense switch 1:

Switch 1 OFF (down) - fixed point format
Switch 1 ON (up) - floating point format

and on the device designated by sense switch 0:

Switch 0 OFF (down) - typewriter
Switch 0 ON (up) - printer.

The program outputs up to seven numbers per line in floating point format and eight numbers per line in fixed point format. This number (seven or eight) is the maximum number of arguments of the TAB instruction which are processed at once. The program cycles, taking the maximum number of arguments at a time until all the arguments have been processed. On each cycle, the location and length of each operand is obtained and saved. When all or the maximum number of operands have been located, the program finds the length of the longest operand. This length is the number of lines TAB will output in the current cycle. The program formats each line, taking the corresponding element from each variable to output. When all the elements of a variable have been output, blanks are output in succeeding lines.

To format a line, program TAB transforms each floating point number to the corresponding string of characters in the proper format. The individual characters are output in EBCDIC code. For each of the two formats, the transformation is designed to make the output of TAB identical to that obtained by a normal FORTRAN write statement. To make the transformation, program TAB uses a character table and a round off control table on the system control file {22.3}.

The user may skip or terminate the execution of the TAB instruction using sense switch 13:

Switch 13 OFF (down) - normal execution
Switch 13 ON (up) - terminate execution.

Program TAB monitors the status of sense switch 13 and suppresses execution as required. Whether the TAB instruction is executed or not, the current location {4-1} is updated by TAB to point to the word in KODE containing the last operand for the TAB instruction.

4.7 PROGRAM LST

Program LST, shown in Figure 4-20, executes the user utility commands: LIST program name, LIST ALL, and EXPLAIN system label. The type of entry to program LST {3.10-1} is set by subsystem 2 to indicate the command to be executed

+ - LIST program name - set to the number of records on the user program file {21.} occupied by the statement count, statement index table, and the source statements of the program

0 - EXPLAIN system label

- - LIST ALL.

For the LIST program function, the FORTRAN record control {7.} has been set to the record on the user program file at which the statement count is stored for the program.

The LIST control {3.9-2} contains the beginning record number on the user program file for the internal form of the program to be listed. For EXPLAIN, the control {3.9-1} points to the row in the system explanation table {22.7} on the system control file {22.} corresponding to the label to be explained. The user has control over the program through the following settings of sense switches on the 1130 console keyboard:

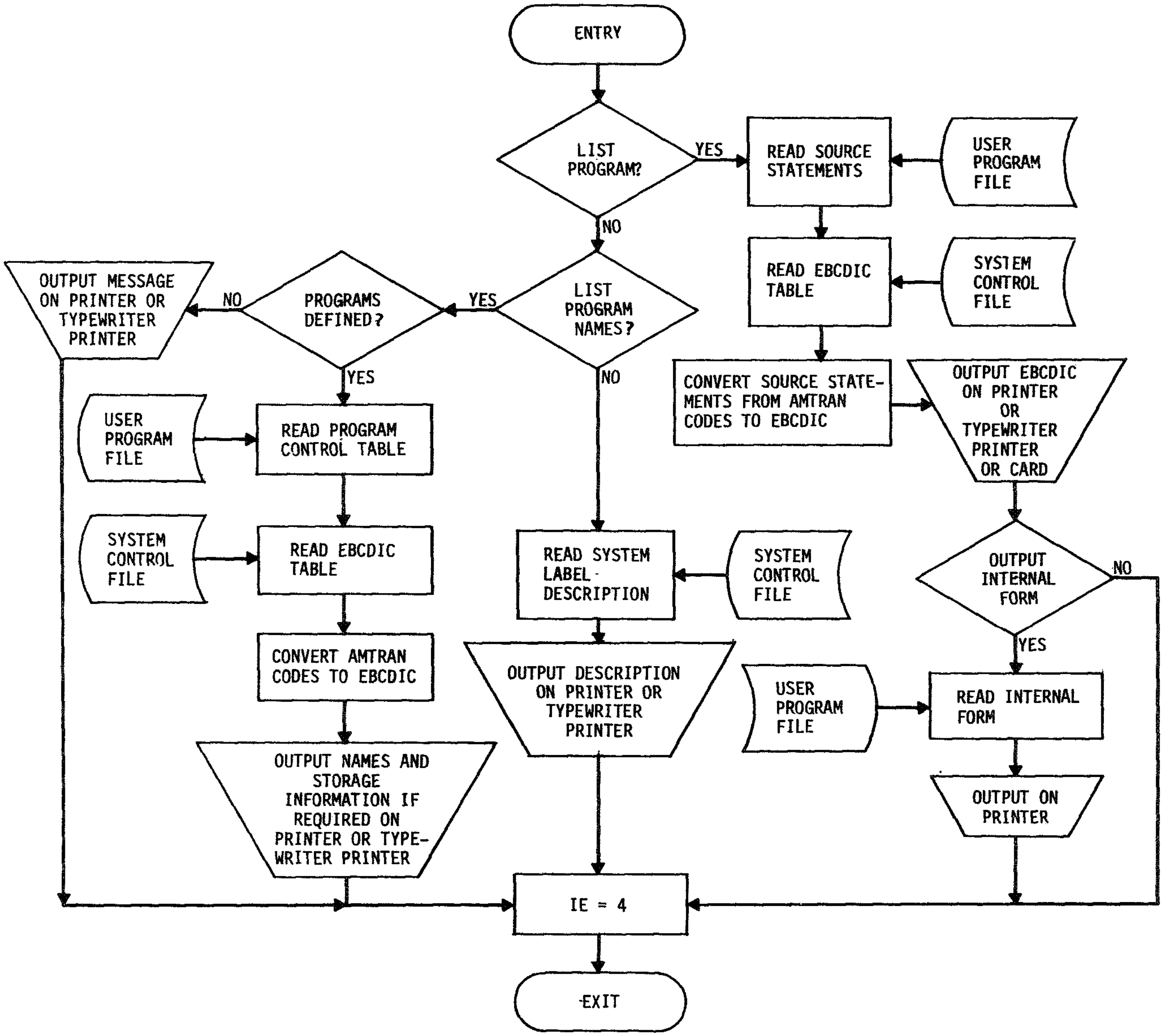

ENTRY
LIST PROGRAM?
YES
NO
READ SOURCE STATEMENTS
USER PROGRAM FILE
READ EBCDIC TABLE
SYSTEM CONTROL FILE
CONVERT SOURCE STATEMENTS FROM AMTRAN CODES TO EBCDIC
OUTPUT EBCDIC ON PRINTER OR TYPEWRITER PRINTER OR CARD
OUTPUT INTERNAL FORM
NO
YES
READ INTERNAL FORM
USER PROGRAM FILE
OUTPUT ON PRINTER
LIST PROGRAM NAMES?
YES
NO
PROGRAMS DEFINED?
NO
YES
OUTPUT MESSAGE ON PRINTER OR TYPEWRITER PRINTER
READ PROGRAM CONTROL TABLE
USER PROGRAM FILE
READ EBCDIC TABLE
SYSTEM CONTROL FILE
CONVERT AMTRAN CODES TO EBCDIC
OUTPUT NAMES AND STORAGE INFORMATION IF REQUIRED ON PRINTER OR TYPEWRITER PRINTER
READ SYSTEM LABEL DESCRIPTION
SYSTEM CONTROL FILE
OUTPUT DESCRIPTION ON PRINTER OR TYPEWRITER PRINTER
IE = 4
EXIT

FIGURE 4-20. PROGRAM LST

Sense Switch	Command(s) Affected	Settings
0	LIST, LIST ALL, EXPLAIN.	OFF - output on typewriter printer ON - output on printer
2	LIST program	OFF - internal form not listed ON - internal form listed on printer
	LIST ALL	OFF - file storage information not output ON - file storage information output with program names
12	LIST program	OFF - listing only ON - program source statements punched on cards
13	LIST program, LIST ALL, EXPLAIN	OFF - normal output ON - output suppressed or terminated if in process.

The actions taken by LST to perform each of the three operations are described below. After completion of the output functions, the error indicator {9.} is set to nonzero so that the statement count {18.} will not be incremented. The system flow goes to subsystem 2 for input of a new source statement.

4.7.1 LIST Program Name

To list a program, the source statements are read from the user program file. Each word contains two AMTRAN coded characters. The characters are unpacked and converted to EBCDIC codes using the EBCDIC table on the system control file {22.4}. As each carriage return or end of statement is encountered, an unpacked line is output on the selected device. If requested, the internal form of the program is read from the file and output with the interpreter instructions unpacked into the operator and operand codes.

4.7.2 EXPLAIN System Label

The row in the system explanation table {22.7} specified by the EXPLAIN control {3.9-1} is read from the system control file. This provides the record number on the file at which the explanation begins and the number of records in the explanation. The records containing the explanation are then read from the file {22.8}. The explanation is output in A2 format one line at a time. Lines are separated in the stored explanation by a word containing the special symbols $$ (the integer equivalent is 23387).

4.7.3 LIST ALL

The user program table {21.4} is read from the user program file. The six characters in each program name are unpacked and converted from AMTRAN to EBCDIC codes. The labels are output in EBCDIC on the selected device with the program storage information, if requested.

4.8 SERVICE SUBPROGRAMS

4.8.1 Subprogram KYBRD

The Assembler language subprogram KYBRD, shown in Figure 4-21, is called by programs RDLL (subsystem 1) and STV (subsystem 4) to provide the link from these FORTRAN programs to the Assembler language subprogram TYPAM which cannot be directly called from FORTRAN and which reads the console keyboard. Subprogram KYBRD is called with one parameter which is the beginning location of the FORTRAN array into which subprogram TYPAM is to place the input statement. Subprogram KYBRD places this parameter in the machine accumulator before calling TYPAM.

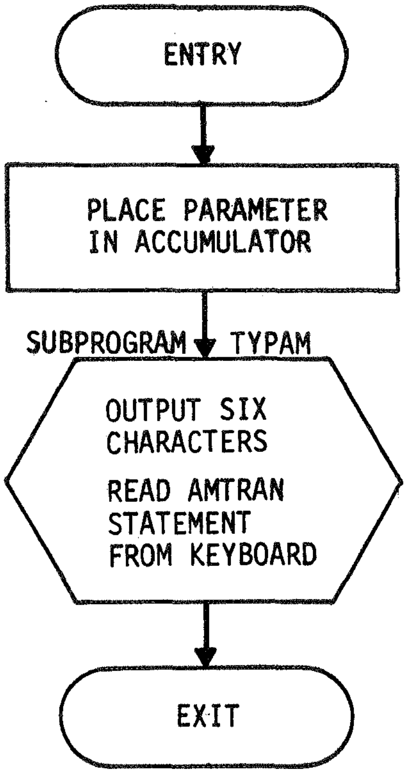

ENTRY
PLACE PARAMETER IN ACCUMULATOR
SUBPROGRAM TYPAM
OUTPUT SIX CHARACTERS
READ AMTRAN STATEMENT FROM KEYBOARD
EXIT

FIGURE 4-21. SUBPROGRAM KYBRD

4.8.2 Subprogram TYPAM

The Assembler language subprogram TYPAM, shown in Figure 4-22, reads a complete AMTRAN source statement from the console keyboard and outputs the statement on the console printer. Subprogram TYPAM, when executed, temporarily replaces the normal interrupt service routine for the keyboard/printer. On entry to TYPAM, the interrupt branch address for interrupt level 4 (core location 12) is replaced with an entry point into TYPAM. The original address is restored before an exit from TYPAM.

The beginning location of the array into which the statement is to be read is passed to TYPAM in the machine accumulator from subprogram KYBRD. Subprogram TYPAM first outputs the first six characters in this array and then reads the statement into the remainder of the array. (If the subprogram is being called indirectly from program RDLL, the six characters are the statement number; if the call is from program STV, the characters are blanks.) The statement is placed in this array as a string of AMTRAN coded characters, one character per word. One character at a time is read and checked for the control characters:

- Carriage return: the ← button on the keyboard
- End of statement: the EOF button on the keyboard.

If the character is not one of these, it is output on the console typewriter and placed in the output array in AMTRAN character code (see Table 4-2). If the character is one of the following, the action described is taken:

- Backspace (the # button on the keyboard) - remove the backspace and the preceding character from the output array

- Delete (the $ button on the keyboard) - remove the $ character from the output array.

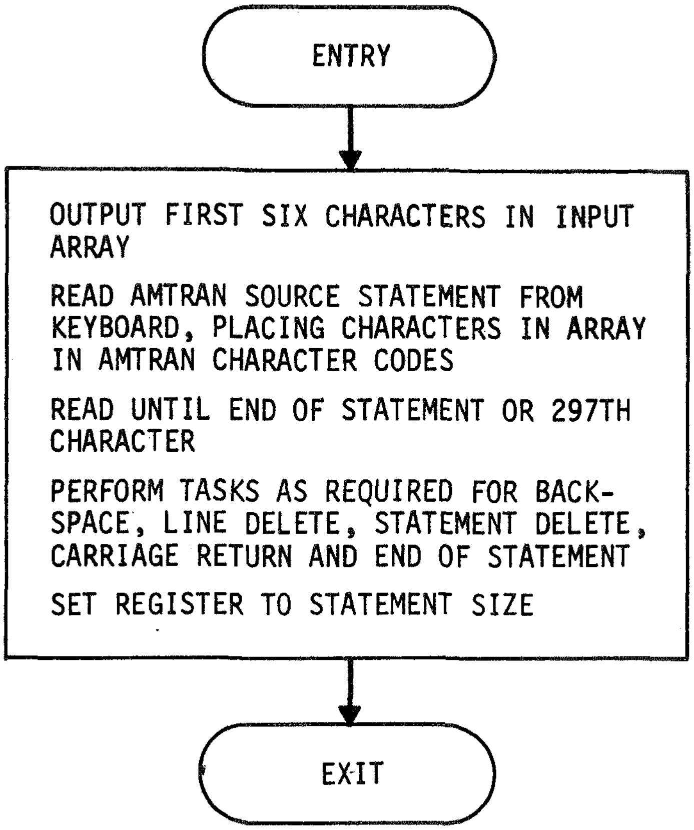

ENTRY
OUTPUT FIRST SIX CHARACTERS IN INPUT ARRAY
READ AMTRAN SOURCE STATEMENT FROM KEYBOARD, PLACING CHARACTERS IN ARRAY IN AMTRAN CHARACTER CODES
READ UNTIL END OF STATEMENT OR 297TH CHARACTER
PERFORM TASKS AS REQUIRED FOR BACKSPACE, LINE DELETE, STATEMENT DELETE, CARRIAGE RETURN AND END OF STATEMENT
SET REGISTER TO STATEMENT SIZE
EXIT

FIGURE 4-22. SUBPROGRAM TYPAM

On the carriage return and the end of statement, the program determines if a $ or a $$ sequence has been entered in the current line, but not backspaced out, or if a backspace has been entered. If a $ has been entered, indicating a delete line action, the current line (except for the first six characters) is deleted from the output array and the program outputs the first six characters in the line and rereads the line from the keyboard. If a $$ has been entered, indicating a delete statement action, the entire statement is deleted from the array and the program types the first six characters again and reads the new statement. If a backspace has been entered on the current line, the corrected line is output on the typewriter and subprogram TYPAM continues to read the input line. If none of these conditions exists when a carriage return is entered, the following tasks are done:

- Enter a carriage return and six blanks in output array
- Output a carriage return and six blanks to the console typewriter
- Read a new input line.

The actions caused by the entry of a carriage return are also taken when the seventy-fourth character has been entered in a line. When the end of statement is entered and a $, $$, or backspace has not occurred in the current line, the read of a source statement is terminated. Before exiting, program TYPAM places a period at the end of statement in the output array, types a period and sets the working register {5.} to the total length of the statement. The actions caused by the entry of the end of statement are also taken when the output array contains 297 characters. However, in this case, the final period is not placed in the output array or typed.

4.8.3 Subprogram SERCH

The Assembler language subprogram SERCH, called only by program SCA (subsystem 2) and shown in Figure 4-23, is used to

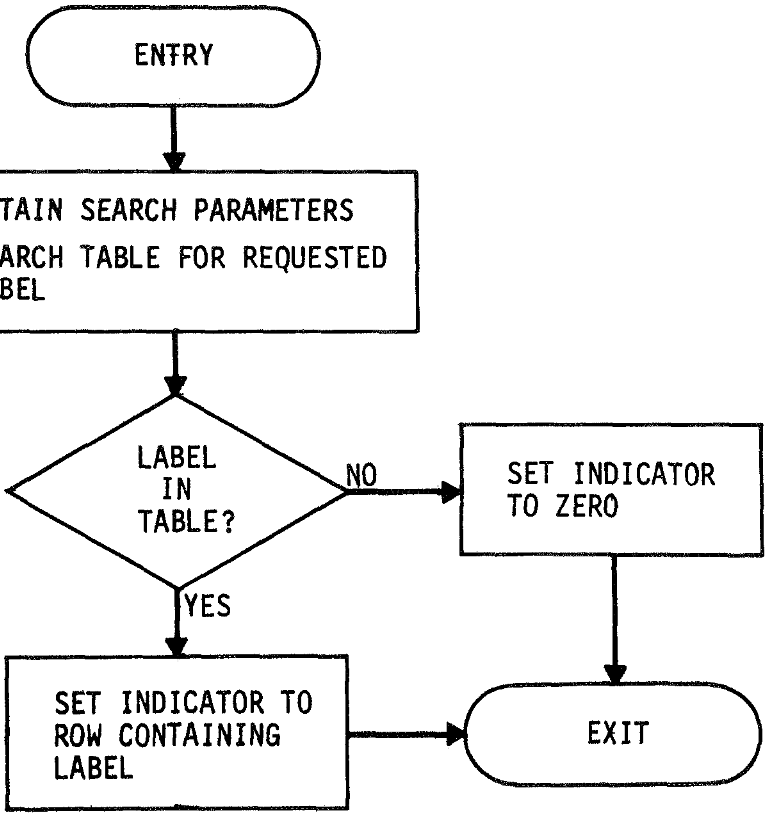


FIGURE 4-23. SUBPROGRAM SERCH

search for a particular label in either the variable table {1.2-1.5}, the system label table {22.1}, or the user program table {21.4}. The following arguments are provided through the parameter string to SERCH in the order listed:

- The location in core of the beginning of the table to be searched
- The number of rows in the table
- The number of rows to be searched, beginning with the first row.

The label to be located in the particular table is always provided in three words in COMMON, beginning with KODE (737). The label consists of six characters packed two AMTRAN coded characters per word. The labels in each of the tables are in the same format and occupy the first three columns of each table. (The tables are stored in core by columns.) Subprogram SERCH searches the indicated table until either the requested label is found or the prescribed number of rows have been searched. If the label is located in the table, the working register {6.} is set to the number of the row in which the label is entered. Otherwise, the register is set equal to zero.

4.8.4 Subprogram AJS

Subprogram AJS, shown in Figure 4-24, makes changes in the storage allocation for data upon request from the programs in subsystem 4 and program DLT in subsystem 1. The subprogram is called with three parameters which contain the following information:

Parameter	Contents
1	Current location of contiguous storage block: subscript of beginning location in a floating point array equivalenced to KODE

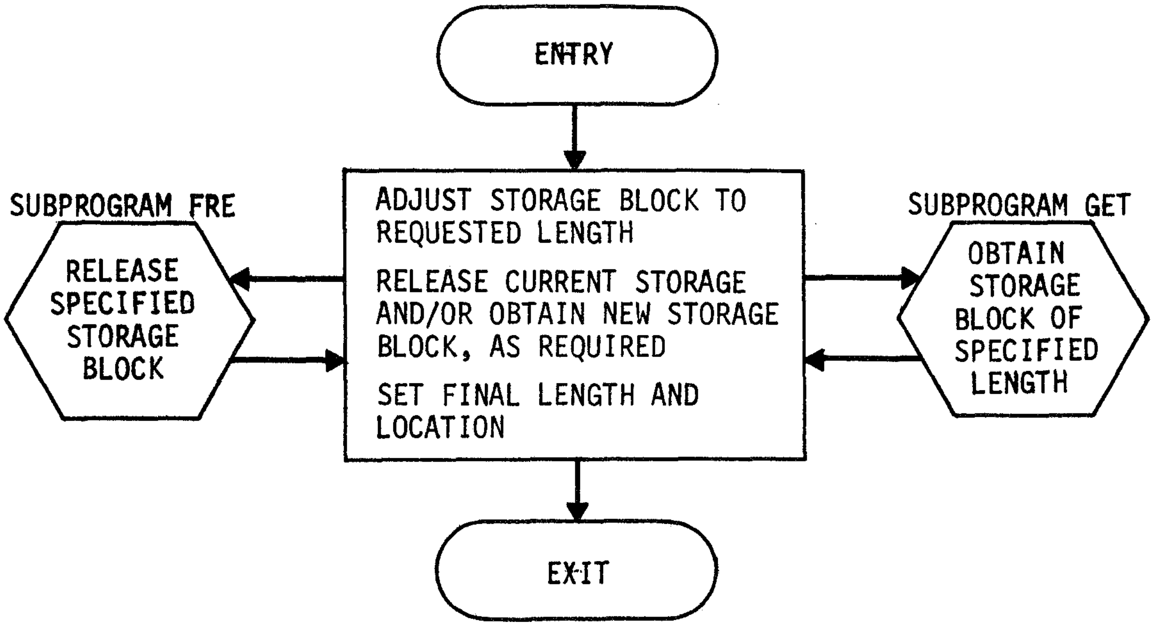

ENTRY
ADJUST STORAGE BLOCK TO REQUESTED LENGTH
RELEASE CURRENT STORAGE AND/OR OBTAIN NEW STORAGE BLOCK, AS REQUIRED
SET FINAL LENGTH AND LOCATION
SUBPROGRAM FRE
RELEASE SPECIFIED STORAGE BLOCK
SUBPROGRAM GET
OBTAIN STORAGE BLOCK OF SPECIFIED LENGTH
EXIT

FIGURE 4-24. SUBPROGRAM AJS

Parameter	Contents
2	Current length of the block: number of floating point words
3	Requested length of the block: number of floating point words.

The subprogram may be called with parameters 1 and 2 equal to zero, indicating a new block is to be obtained. The third parameter is zero if the entire block is to be released. Also, the current and requested lengths may be the same.

Subprogram AJS adjusts the block to the requested length which may involve both releasing storage and obtaining new storage. The first two parameters are changed, respectively, to the following information prior to exiting from AJS:

Parameter	Contents
1	Final location of storage block: subscript of beginning location in a floating point array equivalenced to KODE
2	Final length of storage block: number of floating point words.

The subprogram does not move any data.

4.8.5 Subprogram GET

Subprogram GET obtains a block of specified length from the available storage in the user data area {1.3.1}. The length of the contiguous block to be obtained is provided in the allocation parameter {3.10-2}. The length is the number of floating point words required. Subprogram GET sets an output parameter {3.9-3} to one of the following values:

< 0 - error in storage allocation

0 - a block of the requested length is not available

>0 - the subscript in KODE at which the block of the requested length begins.

The information specifying what storage in the data area is currently available is maintained by the data storage count {12.} and a linkage system in the data area. Each link is two integer words which contain the following information:

Word 1 - location of word 1 of next link (subscript in KODE)

Word 2 - number of contiguous floating point words in the block beginning with the next link pair; the number includes the link as one floating point word.

The first link is always stored in KODE (1141) and KODE (1142). The last link in the chain always contains zeros. The links are ordered in the data area, so that each link points to a link in a higher subscript location. The linkage is initialized to reflect the initial condition of one 604 floating point word block in the data area.

The functions of subprogram GET are described as four tasks which are shown in Figure 4-25. Unless otherwise stated, the tasks are executed in the order in which they are presented.

4.8.5.1 Task 1 - Subprogram GET checks the data storage count {12.} to make sure the requested storage is available.

4.8.5.2 Task 2 - Subprogram GET works through the linkage chain to find the first storage block of at least the requested size. When such a block is located, subprogram GET adjusts the current linkage to remove either the entire block or the requested number of words from the beginning of the block. The subprogram sets the output parameter {3.9-3}, decrements the data storage count {12.} by the number of words removed from the available storage, and returns to the calling program.

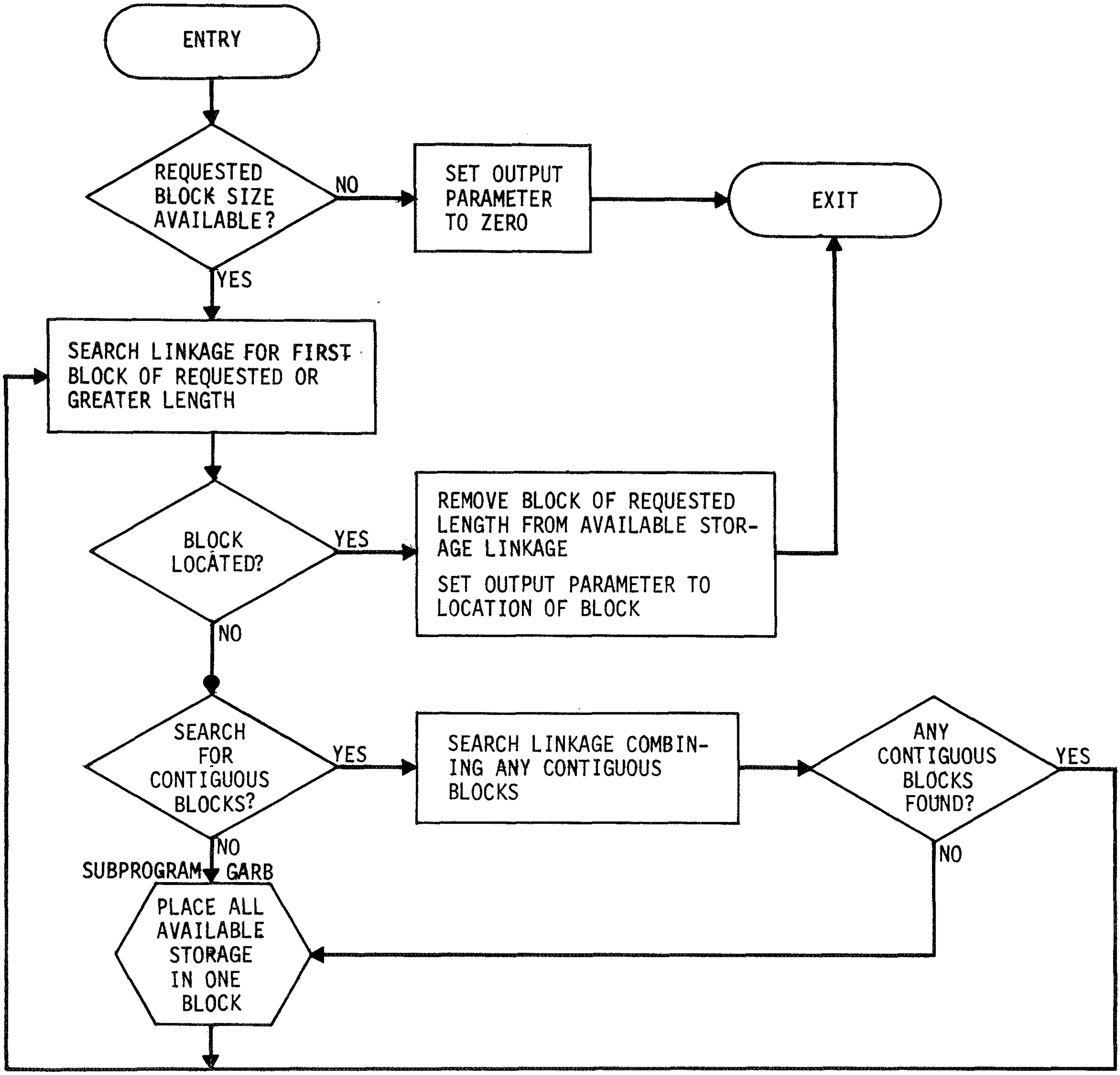

ENTRY
REQUESTED BLOCK SIZE AVAILABLE?
NO
SET OUTPUT PARAMETER TO ZERO
EXIT
YES
SEARCH LINKAGE FOR FIRST BLOCK OF REQUESTED OR GREATER LENGTH
BLOCK LOCATED?
YES
REMOVE BLOCK OF REQUESTED LENGTH FROM AVAILABLE STOR-AGE LINKAGE
SET OUTPUT PARAMETER TO LOCATION OF BLOCK
NO
SEARCH FOR CONTIGUOUS BLOCKS?
YES
SEARCH LINKAGE COMBIN-ING ANY CONTIGUOUS BLOCKS
ANY CONTIGUOUS BLOCKS FOUND?
YES
NO
NO
SUBPROGRAM GARB
PLACE ALL AVAILABLE STORAGE IN ONE BLOCK

FIGURE 4-25. SUBPROGRAM GET

4.8.5.3 Task 3 - If a block of large enough size is not located, the linkage is again followed, looking for any blocks which are contiguous. When two contiguous blocks are located, the blocks are combined into one block by removing one link and incrementing the word count in the appropriate link. If contiguous blocks do not occur, task 4 is executed next; otherwise, task 2 is executed. If the appropriate storage is not located by task 2, task 4 is then performed.

4.8.5.4 Task 4 - Subprogram GARB is called to move all currently allocated storage into one contiguous area and, thereby, provide one block of available storage of length equal to the data storage count {12.}. Task 2 is then performed.

4.8.6 Subprogram GARB

The Assembler language subprogram GARB, called only by subprogram GET, moves all currently used storage in the user data area {1.3.1} into one contiguous block beginning at KODE (1145) and, thereby, provides one block of available storage at the end of the data area and of length equal to the current value of the data storage count {12.}. Subprogram GARB is shown in Figure 4-26.

The data table {2.} is used to control the movement of data blocks. A search of the currently used rows of the data table is made to find any rows with zero entries in the first column. The number of rows is specified by the data table entry count {10.}. The row reserved for the accumulator is also included in the processing. Each of the zero entry rows is counted as already processed. The remainder of the rows are then processed using the following algorithm where the temporary count is first initialized to 573.

- Locate that entry in the data table which has the smallest value in column one greater than or equal to the temporary count.

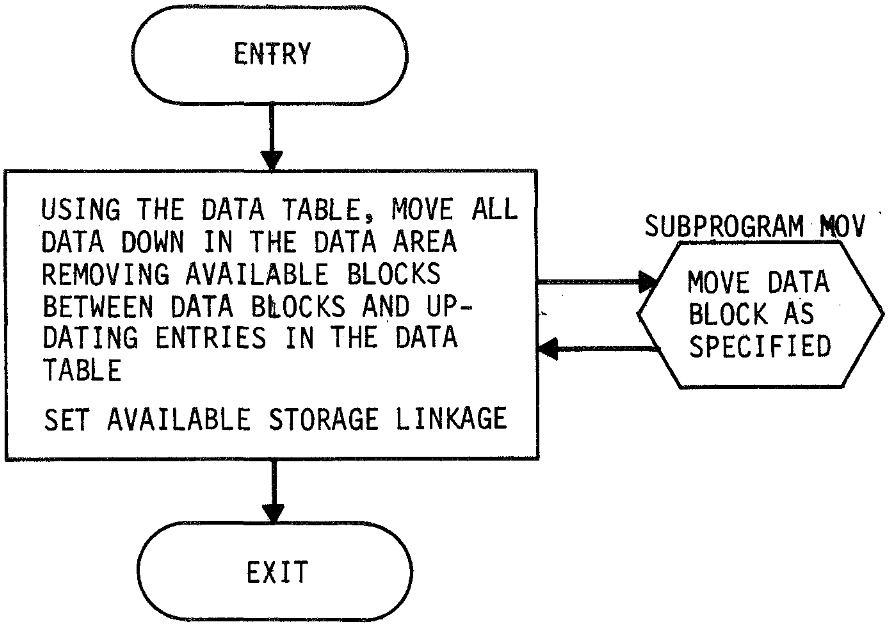

ENTRY
USING THE DATA TABLE, MOVE ALL DATA DOWN IN THE DATA AREA REMOVING AVAILABLE BLOCKS BETWEEN DATA BLOCKS AND UPDATING ENTRIES IN THE DATA TABLE
SET AVAILABLE STORAGE LINKAGE
SUBPROGRAM MOV
MOVE DATA BLOCK AS SPECIFIED
EXIT

FIGURE 4-26. SUBPROGRAM GARB

- For this entry, move the number of floating point data words specified by column two from the location specified by column one to the location specified by the temporary count.

- Set column one to the temporary count.

- Increment the temporary count to the first word after the moved data.

The first link, KODE (1141) and KODE (1142) (see Section 4.8.5 for an explanation of the available storage linkage), is set to the following:

KODE (1141) - subscript in KODE which is the equivalent of the final value of the temporary count

KODE (1142) - total number of floating point words available.

The second link (the location is specified by the first link) is set to zeros.

4.8.7 Subprogram FRE

Subprogram FRE is called to release storage in the user data area {1.3.1} for further use. The two input parameters {3.9-3} and {3.10-2} provide the following:

{3.9-3} - the subscript location in a floating point array equivalenced to KODE at which the block to be released begins

{3.10-2} - the number of floating point words comprising the block.

Subprogram FRE moves along the available storage linkage (see Section 4.8.5) until either the last link or the first link after the block to be released is located. The current linkage is adjusted to include the newly available block.

4.8.8 Subprogram MOV

Subprogram MOV is called to move data within the COMMON array KODE. Three parameters are provided in a call to subprogram

MOV. All of the parameters are relative to a floating point array equivalenced to the integer array KODE. The parameters contain the following information:

Parameter	Contents
1	Subscript location to which data is to be moved
2	Subscript at which data currently is stored (subscript of first element)
3	Number of contiguous floating point words to be moved.

Subprogram MOV calculates the integer subscripts (the actual subscripts in KODE) corresponding to the first two parameters and moves each floating point word as two integer words.

4.8.9 Subprogram SHF

Subprogram SHF exchanges entries in the data table {2.}. Subprogram SHF is called with two parameters, each of which specifies a row in the table. The column 1 and column 2 entries in these rows are exchanged.

4.9 FILE INITIALIZATION PROGRAMS

The file initialization programs are each executed independently and must be executed before executing the system.

4.9.1 Program INTL3

Program INTL3 initializes the first three records on the user program file {21.}:

Record Designation	Initial Value
File status {21.1}	0
First available record {21.2}	580
Number of programs {21.3}	0.

4.9.2 Program ALT5

Program ALT5 initializes all of the system control file {22.} except the system explanation areas {22.7} and {22.8}. The program performs five distinct tasks.

4.9.2.1 Task 1 - The AMTRAN character set is read from an input card. The characters are punched one per column, beginning with column one, and are punched in the order in which they are listed in Table 4-2. The characters are read in A1 format: the codes become the EBCDIC table {22.4}.

4.9.2.2 Task 2 - Program ALT5 either initializes or modifies the system label table {22.1}, depending on the status of sense switch 10:

Switch 10 OFF (down) - initialize table
Switch 10 ON (up) - modify table.

If the table is to be initialized, it is first set to contain blank labels: all entries are set to 2570 which is the code for two AMTRAN character coded blanks packed in a single word. If the table is to be modified, the current table is read from the file.

The program then reads cards which contain the following information in the indicated columns:

Columns	Contents
1-3	Integer (right justified) specifying row in which label is to be entered
6-11	Label (left justified)
20-23	Integer (right justified) which is the code to be entered in column 4 of the table .

One card at a time is processed. The label is read in A1 format, converted to AMTRAN character codes using the table obtained in task one, and packed into three words. The label and code are entered into the table in the specified row. The program continues to process cards until a negative or zero row is specified. The completed table is then written on the file. If the table was only modified, the remaining tasks are not performed.

4.9.2.3 Task 3 - The array {22.2} used by program SCB for reformatting a REPEAT statement is initialized to the sequence: 430, 264, 386, 268, 430, 264, 430, 261, 387, 268, 218, 265, 430, 262, and 265. Each code is placed in one word.

4.9.2.4 Task 4 - The array {22.3} used by programs STV and TAB is initialized by setting the first four words, respectively, to 1, 10, 100, and 1000 and by reading the remainder of the information from a card. The card is punched with thirty characters and is read in A1 format. The first seventeen characters are the character subset used by STV followed by the character $. The characters are punched in the order in which they are listed in Table 4-6 and are read into the last seventeen words of the array. The remaining thirteen characters are the sequence 0.00000E 00 - and are placed in order in the array, beginning in word 5. The completed array is written on the disk, followed immediately by the table obtained in task 1.

4.9.2.5 Task 5 - To place the error messages {22.6} and control table {22.5} on file, the program reads each message from one punched card in A2 format into a forty-word array. Beginning with the fortieth word, the program searches backwards until the sequence ** is located in the array. This is the end of the error message. The program places the current record number in the next row of the table. (The record number begins at 601.) The program writes the message through the ** on the file, beginning at the current record number. The record number is then incremented by the length of the message. A blank card will cause an entry to be made in the table; however, nothing with be written on the disk. The program always reads seventy cards. When all cards have been read and the table is complete, the control table {22.5} is written on the disk.

4.9.3 Program ALT5A

Program ALT5A initializes the system explanation control table {22.7} and explanations {22.8} on the system control file {22.}.

The control table is generated as the explanations are read from cards. The table is first cleared, a word count is set to zero, and the file control is set to 2212. Each explanation is punched on consecutive cards. The last card contains a $$ sequence beginning in an odd numbered column to indicate the end of the explanation. Each card is read in using a 40A2 format. Beginning at the end of the card image, the program searches backwards to locate the first nonblank characters. If the nonblank word does not contain the sequence $$, the following steps are done:

- Transfer the characters to a working array beginning at the word count. The trailing blank words are not transferred.
- Place a $$ (integer 23387) in the next word of the working array.
- Set the word count to point to the word in the working array after the $$.

If the card contains only blanks, the entire sequence of blanks is transferred to the working array. When the $$ sequence occurs, all of the preceding words are transferred to the working array and the word count is incremented to the last word occupied in the array. Column one of the control table is set to the current value of the FORTRAN record control. Column two is set to the word count for the explanation. The explanation is written on the file at the record specified by the record control.

Fifty-six explanations are read and processed. The completed control table is then written on the file.

4.10 MODIFIED 1130 LIBRARY SUBROUTINES

To include the semicolon in the AMTRAN character set, it was necessary to modify the library subroutines HOLTB and EBCTB. The routines are the card/keyboard code table and console printer code table, respectively, and are used by the FORTRAN input/output routines. In the tables, the codes for a percent sign were replaced with the codes for a semicolon.

APPENDIX A. ERROR MESSAGES

Table A-1 lists the error messages output by program RTN in subsystem 4. The error number is the value to which the error indicator {9.} is set for each error. Table A-2 shows, for each error number, the programs in the system which detect the particular error and set the error indicator accordingly. Programs AMTRN, CTL, ITZ, RST, KYBRD, TYPAM, and SERCH which do not appear in Table A-2 either do not detect errors or output error messages directly.

TABLE A-1. ERROR NUMBERS AND MESSAGES

Number	Message
1	INCOMPLETE SUBSTATEMENT
2	ILLEGAL FORMAT FOR IF STATEMENT
3	ILLEGAL USE OF LIST OR EXPLAIN
4	ILLEGAL USE OF GO OR TO
5	LEFT AND RIGHT PARENTHESES UNBALANCED
6	SYSTEM ERROR - ILLEGAL CHARACTER IN POLISH STACK
7	ILLEGAL USE OF INTERVALS
8	SYSTEM ERROR - UNBALANCED POLISH STACK
9	SYSTEM ERROR - ADDRESS IN DELIMITER LIST
10	PROGRAM EXCEEDS AVAILABLE CONSTRUCTION SPACE
11	UNDEFINED STATEMENT NUMBER IN GO TO STATEMENT
12	SYSTEM ERROR - EXTRANEOUS VARIABLE WITHOUT OPERATION
13	EXCEEDING CONSOLE PROGRAM STORAGE AREA
14	EXCEEDING MAXIMUM NUMBER OF CONSOLE PROGRAMS
15	THIS NAME HAS BEEN PREVIOUSLY DEFINED
16	CONSOLE PROGRAM NAME MUST EXCEED ONE CHARACTER
17	MULTIPLE DECIMAL POINT IN CONSTANT
18	UNUSED
19	ILLEGAL USE OF EDIT
20	SYSTEM ERROR - ILLEGAL OPERATION CODE
21	ILLEGAL PARAMETER STRING
22	UNDEFINED VARIABLE
23	ILLEGAL SUBSCRIPT
24	ILLEGAL ARRAY PARAMETERS
25	APPLYING RELATIONAL OPERATION TO NONSCALAR OPERANDS
26	SYSTEM ERROR - NEGATIVE ARRAY LENGTH
27	DIMENSION INCOMPATABILITY
28	ATTEMPTING TO STORE IN CONSTANT LOCATION
29	EXCEEDING DATA AREA - x^1 WORDS AVAILABLE
30	SYSTEM ERROR - NEGATIVE OR ZERO TABLE REFERENCE
31	EXCEEDING CAPACITY OF DATA TABLE
32	SYSTEM ERROR - SORTING ERROR IN DELETE
33	EXCEEDING 54 CONSTANTS IN A CONSOLE PROGRAM
34	UNUSED
35	SYSTEM ERROR - NEGATIVE LINK IN DATA STORAGE
36	SYSTEM ERROR - ERROR IN STORAGE ALLOCATION
37	ILLEGAL USE OF SAVE OR DELETE
38	ILLEGAL OPERAND FOR PARTICULAR OPERATION
39	SYSTEM ERROR - NEGATIVE PROGRAM INDEX
40	EXCEEDING 29 VARIABLES IN A CONSOLE PROGRAM
41	ATTEMPTING TO EXIT AT KEYBOARD LEVEL
42	EXCESSIVE NUMBER OF PARAMETERS
43	INSUFFICIENT NUMBER OF PARAMETERS
44	EXCEEDING TEN CONSOLE PROGRAMS CALLED IN A PROGRAM
45	OPERATION CAN NOT HAVE SCALAR OPERAND

TABLE A-1 - Concluded

Number	Message
46	STATEMENT TOO LONG
47	SYSTEM ERROR - NEGATIVE NUMBER OF PARAMETERS
48	ATTEMPTING RECURSIVE CALL TO A CONSOLE PROGRAM
49	SYSTEM ERROR - ERROR IN CORE ALLOCATION
50	UNDEFINED CONSOLE PROGRAM
51	CONSOLE PROGRAMS EMBEDDED MORE THAN TEN LEVELS
52	CONSOLE PROGRAM EXECUTION AREA IS FULL
53	ILLEGAL CHARACTER
54	EXCEEDING 45 STATEMENTS IN A CONSOLE PROGRAM
55	SYSTEM ERROR - ERROR IN EXECUTING TYPEOUT
56	UNUSED
57	STATEMENT INCOMPLETE
58	STATEMENT NUMBERS NOT IN SEQUENCE
59	DUPLICATE STATEMENT NUMBER
60	UNNUMBERED STATEMENT
61	ILLEGAL STATEMENT NUMBER
62	ILLEGAL STATEMENT
63	ILLEGAL USE OF NAME
64	ILLEGAL USE OF INTRINSIC
65	ILLEGAL SHIFT PARAMETERS
66	ILLEGAL USE OF TYPEOUT
67	UNUSED
68	UNUSED
69	LABEL TOO LONG

[1] The number of words available, x, is provided when the message is output.

TABLE A-2. ERROR MESSAGES REQUESTED BY PROGRAMS IN THE SYSTEM

Error Number Program	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
NAM										X	X		X	X							X				
EDT																			X						
DLT																									
RDLL										X															
SCA			X	X											X	X			X						
SCB		X		X	X						X						X								
STK	X					X		X	X												X				
CDR										X	X	X									X		X	X	
GETOP																				X					
RTN																									
JMP																									
STV																									
WRT																						X			
LSG							X													X		X	X		X
TRG																						X		X	
TAB																						X			
LST																									
AJS																									
GET																									
GARB																									
FRE																									

TABLE A-2 - Continued

Error Number / Program	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
NAM																				
EDT					X							X								
DLT					X		X													
RDLL					X															
SCA					X										X				X	
SCB					X			X				X								
STK					X															
CDR					X															
GETOP																				
RTN					X									X		X				
JMP					X	X											X	X		
STV				X	X	X							X							
WRT					X								X							
LSG	X	X	X		X	X							X	X						
TRG	X	X			X								X					X		X
TAB					X								X							
LST													X							
AJS					X															
GET	X			X						X										
GARB					X															
FRE										X	X									

TABLE A-2 - Concluded

Error Number / Program	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69
NAM																		X						
EDT									X							X								
DLT																								
RDLL	X							X	X			X	X	X	X									
SCA					X													X			X			X
SCB	X															X	X							
STK	X																X		X					
CDR	X																X			X				
GETOP																								
RTN																								
JMP		X	X		X	X	X																	
STV																								
WRT										X											X			
LSG																								
TRG																								
TAB																				X				
LST																								
AJS																								
GET																								
GARB				X																				
FRE																								

APPENDIX B. PROGRAM LISTINGS

Listings for the programs in the system appear in this appendix in the order in which the programs are described in Section 4. An alphabetized index is provided on following page. Although the system operates only under Version II of the 1130 Disk Monitor System, because of core limitations the FORTRAN programs must be compiled under Version I. The two Assembler language programs CTL and GETOP must be assembled using Version II; however, the remaining Assembler language programs can be assembled using either version.

Index to Program Listings

PROGRAM AMTRN

```
*LIST ALL
*IOCS(CARD,1132 PRINTER,TYPEWRITER,DISK)
*ONE WORD INTEGERS
*LIST SOURCE PROGRAM
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      DEFINE FILE 1(7360,1,U,II)
      DEFINE FILE 3(30720,1,U,II)
      DEFINE FILE 5(14080,1,U,II)
    1 CALL CTL
      GO TO 1
      END
```

PROGRAM CTL

```
      ENT       CTL                                          CTL00010
KODE  EQU       /7FFF                                        CTL00020
KOP   EQU       /75FF                                        CTL00040
KOA   EQU       /75FE                                        CTL00050
KP    EQU       /75F9                                        CTL00060
I     EQU       /75F5                                        CTL00070
J     EQU       /75F4                                        CTL00080
L     EQU       /75F3                                        CTL00090
ID    EQU       /75F1                                        CTL00100
IE    EQU       /75F0                                        CTL00110
LPV   EQU       /75EC                                        CTL00120
IEX   EQU       /75E9                                        CTL00130
IGT   EQU       /75E8                                        CTL00140
CTL   DC        0                                            CTL00150
      CALL      ITZ                                          CTL00160
L012  CALL      RST                                          CTL00190
L011  CALL      RDLL                                         CTL00200
      LD    L   IE                                           CTL00210
      BNZ       L919                                         CTL00220
L099  CALL      SCA                                          CTL00230
      DC        LR                                           CTL00240
      LDX   I2  LR                                           CTL00250
      BSC   I2  SW1-1                                        CTL00260
SW1   DC        L100                                         CTL00270
      DC        L012                                         CTL00280
      DC        L115                                         CTL00290
      DC        L118                                         CTL00300
      DC        L119                                         CTL00310
      DC        L919                                         CTL00320
      DC        L120                                         CTL00335
L115  CALL      LST                                          CTL00360
      B         L011                                         CTL00370
L100  CALL      SCB                                          CTL00380
      DC        LR                                           CTL00390
      LD        LR                                           CTL00400
      S         TWO                                          CTL00410
      BN        L101                                         CTL00420
      BNZ       L919                                         CTL00430
L120  CALL      DLT                                          CTL00440
CHKIE LD    L   IE                                           CTL00450
      BNZ       L919                                         CTL00460
      B         L012                                         CTL00470
L118  CALL      NAM                                          CTL00480
      B         CHKIE                                        CTL00490
L119  CALL      EDT                                          CTL00500
      B         CHKIE                                        CTL00510
L101  CALL      STK                                          CTL00520
      LD    L   IE                                           CTL00530
      BNZ       L919                                         CTL00540
L102  CALL      CDR                                          CTL00550
      LD    L   IE                                           CTL00560
      BNZ       L919                                         CTL00570
L201  LD    L   IEX                                          CTL00580
      S         ONE                                          CTL00590
      BNZ       L011                                         CTL00600
L300  LD    L   LPV                                          CTL00610
      A         ONE                                          CTL00620
      STO   L   I                                            CTL00630
      LD        ZRO                                          CTL00640
      STO   L   IGT                                          CTL00650
      B         L402                                         CTL00660
```

```
ONE    DC        1                                                     CTL00670
LR     DC        0                                                     CTL00680
ZRO    DC        0                                                     CTL00690
TWO    DC        2                                                     CTL00700
C3     DC        3                                                     CTL00710
L320   CALL      WRT                                                   CTL00720
L400   LD     L  IE                                                    CTL00730
       BN        L919                                                  CTL00740
       BP        L404                                                  CTL00750
L321   LD        ZRO                                                   CTL00760
       STO    L  IGT                                                   CTL00770
L401   MDM    L  I,1                                                   CTL00780
       LD     L  ID                                                    CTL00790
       S         ONE                                                   CTL00800
       BN        L919                                                  CTL00810
       BP        L301                                                  CTL00820
L402   LD     L  L                                                     CTL00830
       S      L  I                                                     CTL00840
       BN        L011                                                  CTL00850
L301   CALL      GETOP                                                 CTL00860
       LD     L  IE                                                    CTL00870
       BNZ       L404                                                  CTL00880
       LDX    I2 J                                                     CTL00890
       BSC    I2 SW2-1                                                 CTL00900
SW2    DC        L405                                                  CTL00910
       DC        L307                                                  CTL00920
       DC        L309                                                  CTL00930
       DC        L320                                                  CTL00940
       DC        L317                                                  CTL00950
       DC        L318                                                  CTL00960
       DC        L3061                                                 CTL00970
       DC        TAB1                                                  CTL00972
TAB1   CALL      TAB                                                   CTL00974
       B         L400                                                  CTL00976
L3061  WAIT                                                            CTL00980
       B         L321                                                  CTL00990
L307   CALL      JMP                                                   CTL01000
       B         L400                                                  CTL01010
L309   CALL      STV                                                   CTL01020
       B         L400                                                  CTL01030
L318   CALL      TRG                                                   CTL01040
       B         L400                                                  CTL01050
L317   CALL      LSG                                                   CTL01060
       DC        LR                                                    CTL01070
       LDX    I2 LR                                                    CTL01080
       BSC    I2 SW3-1                                                 CTL01090
SW3    DC        L321                                                  CTL01100
       DC        L401                                                  CTL01110
       DC        L012                                                  CTL01120
       DC        L404                                                  CTL01130
L404   LD        TWO                                                   CTL01140
       STO    L  KP                                                    CTL01150
       LD     L  ID                                                    CTL01160
       S         ONE                                                   CTL01170
       BP        L405                                                  CTL01180
L919   LD        C3                                                    CTL01190
       STO    L  KP                                                    CTL01200
L405   CALL      RTN                                                   CTL01210
       DC        LR                                                    CTL01220
```

```
      LDX  I2 LR                                             CTL01230
      BSC  I2 SW4-1                                          CTL01240
SW4   DC      L321                                           CTL01250
      DC      L011                                           CTL01270
      DC      L012                                           CTL01280
      END                                                    CTL01290
```

PROGRAM ITZ

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE ITZ
      DIMENSION KCLA(1),KCLB(1)
      DIMENSION ICD(1)
      DIMENSION IO(2),IW(2),ICPT(95,6)
      DIMENSION DATA(1)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (ICD(1),KODE(841))
      EQUIVALENCE (IW(1),KODE(704)),(IO(1),KODE(1)),(ISF,KODE(1)),
     1(IFR,KODE(2)),(NP,KODE(3)),(ICPT(1,1),KODE(4))
      EQUIVALENCE (DATA(1),KODE(1142))
      EQUIVALENCE (KCLA(1),KODE(972)),(KCLB(1),KODE(1022))
      II=1
      READ (3'II)ISF,IFR,NCP
      CALL DATSW(2,ISS)
      GO TO(40,42),ISS
   40 WRITE (3,41)ISF,IFR,NCP
   41 FORMAT(15I6)
   42 IF(ISF)1,19,1
    1 IF(NCP)100,11,2
    2 II=4
      READ(3'II)((ICPT(J,K),K=1,6),J=1,NCP)
      GO TO(43,44),ISS
   43 WRITE(3,41)((ICPT(J,K),K=1,6),J=1,NCP)
   44 NN=0
      IFR=574
    3 J=0
      DO 7 K=1,NCP
      IF(ICPT(K,4)-IFR)7,10,4
    4 IF(J)6,6,5
    5 IF(ICPT(J,4)-ICPT(K,4))7,6,6
    6 J=K
    7 CONTINUE
      IP=ICPT(J,4)
      NM=ICPT(J,5)-IP
      NRI=NM+ICPT(J,6)
      ICPT(J,4)=IFR
   70 NR=NRI
      IF(NR-1600)72,72,71
   71 NR=1600
   72 II=IP
      READ(3'II)(IW(N),N=1,NR)
      IP=II
      II=IFR
      WRITE(3'II)(IW(N),N=1,NR)
      IFR=IFR+NR
      NRI=NRI-NR
      IF(NRI)73,73,70
   73 ICPT(J,5)=ICPT(J,4)+NM
      GO TO 75
   10 IFR=IFR+ICPT(K,5)-ICPT(K,4)+ICPT(K,6)
   75 NN=NN+1
      IF(NN-NCP)3,12,12
  100 NCP=0
   11 IFR=574
   12 IF(NCP-95)121,15,15
  121 M=NCP+1
      DO 14 K=M,95
      DO 13 N=1,6
   13 ICPT(K,N)=0
   14 CONTINUE
  15  ISF=0
      II=1
      WRITE(3'II)ISF,IFR,NCP,((ICPT(J,M),M=1,6),J=1,95)
      GO TO (45,19),ISS
   45 WRITE (3,41)ISF,IFR,NCP
      WRITE(3,41)((ICPT(J,K),K=1,6),J=1,NCP)
   19 DATA(607)=0.0
      DATA(608)=1.0
      DATA(609)=2.0
      DATA(610)=3.0
      DATA(611)=4.0
      DATA(612)=5.0
      DATA(613)=6.0
      DATA(614)=7.0
      DATA(615)=8.0
      DATA(616)=9.0
      DATA(617)=10.0
      DATA(618)=3.1415927
      DATA(619)=57.2958
      DATA(620)=0.0174533
      KCLA(1)=9
      KCLA(2)=10
      KCLA(3)=49
      KCLA(4)=50
      KCLA(5)=51
      KCLA(6)=208
      KCLA(7)=209
      KCLA(8)=212
      KCLA(9)=218
      KCLA(10)=430
      KCLA(11)=713
      KCLB(1)=1
      KCLB(2)=2
      KCLB(3)=3
      KCLB(4)=4
      KCLB(5)=2
      KCLB(6)=6
      KCLB(7)=5
      KCLB(8)=6
      KCLB(9)=7
      KCLB(10)=6
      KCLB(11)=8
      II=501
      READ(5'II)(ICD(M),M=1,50)
      KODE(7)=1
      II=6282
      WRITE(1'II)(ICD(M),M=1,50)
      IT(8)=1
      IEX=1
      NV=0
      IT(4)=1
      KODE(6)=0
      II=6191
      WRITE(1'II)(KODE(M),M=750,1140)
      RETURN
      END
```

PROGRAM RST

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE RST
      DIMENSION ICPT(96,6),LOOK(6)
      DIMENSION ICA(2),ICB(2),ICC(2),NVA(2),NVB(2),NVC(2),IDA(2),IDB(2)
      DIMENSION ISP(1),NVAR(50,5)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (NVAR(1,1),KODE(891))
      EQUIVALENCE (NP,KODE(4))
      EQUIVALENCE (NPP,KODE(5))
      EQUIVALENCE (NPV,KODE(6))
      EQUIVALENCE (ICARD,KODE(7))
      EQUIVALENCE (ISP(1),KODE(1141))
      EQUIVALENCE (LOOK(1),KODE(513)),(IT1,IT(1))
      EQUIVALENCE (IT4,IT(4)),(IT8,IT(8)),(IT6,IT(6))
      EQUIVALENCE (ICA(1),ICPT(1,4)),(ICB(1),ICPT(1,5)),(ICC(1),ICPT(1,6
     1)),(NVA(1),NVAR(1,1)),(NVB(1),NVAR(1,4)),(NVC(1),NVAR(1,5)),(IDA(1
     2),IDAT(1,1)),(IDB(1),IDAT(1,2))
      EQUIVALENCE (ISP1,ISP(1)),(ISP2,ISP(2)),(ISP5,ISP(5)),(ISP6,ISP(6)
     1),(K750,KODE(750)),(K1,KODE(1)),(K2,KODE(2)),(K3,KODE(3))
      GO TO(21,1,600,500,1),IT4
  600 NRI=NCP+1
      ICRD=ICARD
      II=1
      READ(3'II)ISF,IFR,NCP
      IF(NCP)33,33,34
   34 READ(3'II)((ICPT(J,K),K=1,6),J=2,NRI)
      DO 36 K=2,NRI
      DO 35 M=1,3
      IF(LOOK(M)-ICPT(K,M))37,35,36
   35 CONTINUE
      IP =K
      GO TO 107
   36 CONTINUE
   33 K=NRI+1
   37 IP =K-1
  107 NM=NRI
      NN=IP
      NR=IP
      GO TO (38,38,31),IEX
   31 ISF=1
      IF(IP -1-IT1)32,203,206
   32 NM=IT1
      GO TO 26
  206 IP=IP+1
      NN=IT1+2
  203 NM=IP
      NR=IT1
      GO TO 26
   38 NCP=NCP+1
   26 DO 27 K=1,6
   27 ICPT(IP ,K)=LOOK(K)
      II=1
      WRITE(3'II)ISF,IFR,NCP
      II=6*NR -2
      WRITE(3'II)((ICPT(J,K),K=1,6),J=NN,NM)
      IT8=2
      NM=1
      NN=IEX-1
     GO TO(22,506,512),IEX
 500 IT4=1
     NM=L
 511 GO TO (506,501,509,501),NM
 501 GO TO (502,504,506),IEX
 502 II=3421
     WRITE(1'II)(KODE(K),K=1,1140),(IT(N),N=1,20),(IT(M),M=22,26)
 503 IEX=2
 504 ICARD=NM-1
 505 NPV=0
     GO TO 21
 506 II=IEX*1170+1081
     READ(1'II) (KODE(K),K=1,1140),(IT(N),N=1,20),(IT(M),M=22,26)
     GO TO(5080,507,508,501),NM
 507 GO TO (502,508),IEX
5080 GO TO(508,5081),ICRD
 508 WRITE(1,901)
 901 FORMAT(1H )
5081 IT8=1
     IE=1
     NAP=0
     GO TO(510,18,18),NM
 509 GO TO(508,506,512),IEX
 512 WRITE(1,902)
 902 FORMAT(10X,8HCONTINUE)
     GO TO 506
 510 GO TO (18,39),NN
  39 IT4=3
   1 II=1
     READ(3'II)ISF,IFR,NCP
1003 IP=1
     IF(ISF)1001,17,1001
1001 IF(NCP)120,120,2
   2 IF(IFR-29000)17,17,204
 204 II=4
     READ(3'II)((ICPT(J,K),K=1,6),J=1,NCP)
     NN=0
     IFR=574
   3 J=0
     DO 7 K=1,NCP
     IF (ICA(K)-IFR)7,10,4
   4 IF(J)6,6,5
   5 IF(ICA(J)-ICA(K))7,6,6
   6 J=K
   7 CONTINUE
     GO TO(8,9),IP
   8 II=4591
     WRITE(1'II)(KODE(M),M=1,1600)
     IP=2
   9 IP=ICA(J)
     NM=ICB(J)-IP
     NR=NM+ICC(J)
     ICA(J)=IFR
  46 NRI=NR
     IF(NRI-1600)48,48,47
  47 NRI=1600
  48 II=IP
     READ(3'II)(KODE(M),M=1,NRI)
```

```
      IP=II
      II=IFR
      WRITE(3'II)(KODE(M),M=1,NRI)
      IFR=IFR+NRI
      NR=NR-NRI
      IF(NR)50,50,46
   50 ICB(J)=ICA(J)+NM
      GO TO 11
   10 IFR=IFR+ICB(K)-ICA(K)+ICC(K)
   11 NN=NN+1
      IF(NN-NCP)3,121,121
  120 IFR=574
  121 ISF=0
      II=1
      GO TO (13,14),IP
   13 WRITE(3'II)ISF,IFR,NCP
      GO TO 17
   14 WRITE(3'II)ISF,IFR,NCP,((ICPT(J,M),M=1,6),J=1,NCP)
      II=4591
      READ(1'II)(KODE(M),M=1,1600)
   17 GO TO(201,170,18,18,201),IT4
  170 IE=1
   18 IF(NMB-1)190,201,190
  190 IT4=1
      RETURN
   22 NPV=0
      ICARD=1
  201 IT4=1
   21 NV=NPV
      IF (NV) 2000,2000,2005
 2000 GO TO (2001,205,205),IEX
 2001 IFT=0
      ISP1=1145
      ISP2=604
      ISP5=0
      ISP6=0
      LNT=604
 2005 GO TO(205,200),IT8
  200 II=6191
      READ(1'II)(KODE(M),M=750,1140)
      IT8=1
  205 ID=1
      NAP=0
      NPP=0
      NP=0
      GO TO (100,110,12),IEX
  100 WRITE(1,1002)
 1002 FORMAT(//2X,13HENTER PROGRAM)
      GO TO 112
  110 GO TO(1100,12,112),ICARD
 1100 WRITE(1,111)
  111 FORMAT(/10X,25HENTER PROGRAM-SUPPRESSED-)
  112 DO 113 K=751,840
  113 KODE(K)=0
   12 KON=0
      K750=46
      K1=252
      K2=302
      K3=410
      LPV=7
      N=NV+1
      DO 19 I=N,30
      KODE(I+252)=0
      NVA(I)=0
      NVB(I)=0
   19 NVC(I)=0
      GO TO (1900,1901,1901),IEX
 1900 DO 191 I=N,90
      IDA(I)=0
  191 IDB(I)=0
 1901 DO 20 I=31,50
      NVC(I)=0
      KODE(I+252)=0
   20 NVB(I)=-1
      L=7
      NMB=0
      IT6=1
      IE=0
      RETURN
      END
```

PROGRAM NAM

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE NAM
      DIMENSION NVAR(50,5),NVA(2),NVB(2)
      DIMENSION NTB(2),KEEP(6)
      DIMENSION KCH(2),ICD(50),KTP(80)
      DIMENSION LGO(2)
      DIMENSION LOOK(1)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (NVAR(1,1),KODE(891))
      EQUIVALENCE (NVA(1),NVAR(1,4)),(NVB(1),NVAR(1,5))
      EQUIVALENCE (NTB(1),KODE(751))
      EQUIVALENCE (LGO(1),KODE(796))
      EQUIVALENCE (IT1,IT(1)),(NPV,KODE(6))
      EQUIVALENCE (LG,KODE(750))
      EQUIVALENCE (LOOK(1),KODE(513)),(KCH(1),KODE(511)),(IF2,KODE(51
     X,(L4,KEEP(4)),(L5,KEEP(5)),(L6,KEEP(6)),(K4,KODE(4)),(IT6,IT(6)
     X(K1,KODE(1)),(K2,KODE(2)),(K3,KODE(3)),(IT4,IT(4)),(IT8,IT(8)),
     X(ICD(1),KODE(841))
      EQUIVALENCE (ICARD,KODE(7))
      EQUIVALENCE (KTP(1),KEEP(1))
      ICRD=ICARD
      NP=K4
      K4=0
      M=LPV+1
      IF(M-128)31,30,30
   30 IF(LG-NMB)33,33,32
   32 LG=NMB
   33 M=M-127
   31 LGO(NMB)=128*LGO(NMB)+M
      GO TO(1001,1001,1),IEX
 1001 IF(NCP-95)1,914,914
    1 L=LPV+1
      IF1=1
      IF(KCH(3)-2816)963,7050,7050
 7050 IF(IF2)705,705,700
  700 IF(IF2/100-4)963,701,963
  701 KODE(L)=28160+IF2
      IF2=IF2-400
      L=L+1
      IF1=2
      IF(NVA(IF2))705,703,705
  703 WRITE (1,704)
  704 FORMAT(66H THE VARIABLE CHOSEN FOR FUNCTIONAL RETURN APPEARS TO
     1 UNDEFINED)
  705 IF(L-252)1000,1000,910
 1000 KODE(L)=512
      NSF=IT6-1
  800 IF(NPV-NV)970,706,970
  706 N=1
      IF(NV)660,660,707
  707 DO 708 M=1,NV
  708 NVB(M)=0
      MM=1
      DO 91 M=7,J
      KV=KCH(M)
      K=KV/100+1
      GO TO (820,968,968,968,890),K
  820 IF(KV-50)821,83,968
 821 IF(KV-43)823,86,824
 823 IF(KV-10)968,91,968
 824 IF(KV-45)87,91,87
  86 IF(MM-1)968,89,968
  87 IF((KV-41)/3-1)968,88,968
  88 IF(MM-4)968,89,968
  89 MM=KV-41
     GO TO 91
  83 GO TO(92,968,92,968,968),MM
 890 GO TO(968,891,968,968,891),MM
 891 MM=4
     K=KV-400
     IF(NVB(K))968,90,968
  90 NVB(K)=N
     N=N+1
  91 CONTINUE
     GO TO 968
  92 K4=N-1
     MN=1
     NN=1
     DO 109 M=1,NV
     IF(NVA(M))100,93,100
  93 IF(NVB(M))94,94,109
  94 GO TO(491,490),IF1
 490 IF(M-IF2)491,108,491
 491 GO TO(95,97),NN
  95 WRITE(1,96)
  96 FORMAT(38H  ** THE FOLLOWING APPEAR UNDEFINED **)
     NN=2
  97 IF(MN-1)972,972,971
 971 KTP(MN)=ICD(47)
     MN=MN+1
 972 DO 98 MM=1,3
     JK=NVAR(M,MM)/256
     KJ=NVAR(M,MM)-256*JK
     IF(JK-10)973,981,973
 973 KTP(MN)=ICD(JK+1)
     MN=MN+1
     IF(KJ-10)974,981,974
 974 KTP(MN)=ICD(KJ+1)
  98 MN=MN+1
 981 IF(MN-74)108,982,982
 982 WRITE(1,99)(KTP(MM),MM=1,MN)
  99 FORMAT(8X,80A1)
     MN=1
     GO TO 108
 100 IF(NVB(M))108,108,109
 108 NVB(M)=N
     N=N+1
 109 CONTINUE
     IF(MN-1)660,660,1090
1090 WRITE(1,99)(KTP(MM),MM=1,MN)
 660 GO TO(659,658),IF1
 658 IF(K4-NVB(IF2))659,968,968
 659 IF(NVA(30))664,664,661
 661 NVB(30)=N
     N=N+1
 664 DO 670 M=31,50
```

```
      IF(NVA(M))671,671,665
 665  NVB(M)=N
 670  N=N+1
 671  NV=N-1
      N=8
6710  IA=KODE(N)/512
      IB=KODE(N)-512*IA
      IF(IA-9)6712,6711,6712
6711  N=N+IB+1
      GO TO 681
6712  IF(IA-1)6714,6713,6714
6713  GO TO(680,6715),IF1
6715  IF(N-L)6716,680,680
6716  K=L-1
      GO TO 675
6714  J=IB-400
      IF (J) 680,680,672
 672  K=J-49
      IF (K) 673,673,674
 673  KODE(N)=KODE(N)+NVB(J)-J
      IF(IA-4)680,679,680
 679  N=N+3
      GO TO 680
 674  MM=0
      IF(K-LG)6741,6740,6740
6740  MM=127
6741  K=LGO(K)-128*(LGO(K)/128)+MM
      IF (K) 911,911,675
 675  KODE(N)=6388+K-N
 680  N=N+1
 681  IF(N-L)6710,6710,682
 682  K1=L
      LL=(L+NV+1)/2+1
      M=2*LL-2
      K2=M
      N=L+1
      DO 10 K=N,M
  10  KODE(K)=0
      N=2*KON+M
      K3=N
      IF(KON)13,13,11
  11  CALL MOV (LL,152,KON)
  13  M=N+4*NP
      IF(NP)16,16,14
  14  CALL MOV ((N+2)/2,206,2*NP)
  16  DO 17 K=5,7
  17  KODE(K)=0
      II=2
      READ(3'II)IFR
      L4=IFR
      II=IFR
      IF(IFR+M+2*NMB+NSF-30721)170,170,913
 170  WRITE(3'II)(KODE(K),K=1,M)
      IFR=IFR+M
      DO 4 K=1,3
   4  KEEP(K)=LOOK(K)
      K1=NMB
      N=NMB+1
      DO 18 K=1,NMB
      KODE(K+1)=NTB(K)
      N=N+1
   18 KODE(N)=LGO(K)/128
      IFN=(IEX-1)*1140
      II=1+IFN
      M=2*NMB+2
      N=NSF+M-1
  102 IF(N-1140)103,103,910
  103 READ(1'II)(KODE(K),K=M,N)
      L6=N
      L5=IFR
      II=IFR
      IFR=IFR+N
      IF(IFR-30720)104,104,913
  104 WRITE(3'II)(KODE(K),K=1,N)
      IT4=3
      DO 5 K=1,6
    5 LOOK(K)=KEEP(K)
      II=2
      WRITE(3'II)IFR
      ICARD=ICRD
      RETURN
  970 IE=7
  963 IE=IE+42
  968 IE=IE+7
  914 IE=IE+1
  913 IE=IE+2
  911 IE=IE+1
  910 IE=IE+10
      RETURN
      END
```

PROGRAM EDT

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE EDT
      DIMENSION KCH(1),LGO(45),NTB(45),NTA(45),LGA(45)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (KCH(1),KODE(511))
      EQUIVALENCE (NTA(1),KODE(751))
      EQUIVALENCE(LGA(1),KODE(796))
      J=I
      IP=II
      NAP=0
      II=IP
      READ(3'II)NT
      READ(3'II)(NTB(K),K=1,NT),(LGO(K),K=1,NT)
      CALL DATSW(7,ISW)
      GO TO (98,101),ISW
   98 WRITE(3,100)(NTB(K),LGO(K),K=1,NT)
  100 FORMAT(2I5)
  101 LM=1
    1 NP=0
      ND=0
      LL=0
      NN=1
    2 J=J+1
      IF(KCH(J)-10)3,2,4
    3 NP=NP*10+KCH(J)
      J=J+1
      IF(KCH(J)-10)3,12,4
    4 IF(KCH(J)-45)919,9,5
    5 IF(KCH(J)-46)919,14,6
    6 IF(KCH(J)-50)919,7,8
    7 IF(NP+ND)13,31,13
    8 IF(NP+ND)14,2,14
    9 J=J+1
      IF(KCH(J)-10)10,12,110
   10 ND=10*KCH(J)
      J=J+1
      IF(KCH(J)-10)11,12,110
   11 ND=ND+KCH(J)
      J=J+1
      IF(KCH(J)-10)961,12,110
  110 IF(KCH(J)-45)919,111,5
  111 J=J+1
      IF(KCH(J)-10)919,111,112
  112 IF(KCH(J)-50)919,13,111
   12 NN=NN+1
   13 NN=NN+1
   14 IF(NP-99)16,16,961
   16 NS=256*NP+ND
      IF(NS)961,961,17
   17 IF(NS-NTB(NT))18,18,919
   18 DO 19 K=1,NT
      IF(NS-NTB(K))24,20,19
   19 CONTINUE
      GO TO 919
   20 IF(LGO(K))23,21,23
   21 NT=NT-1
      DO 22 M=K,NT
      LGO(M)=LGO(M+1)
  22 NTB(M)=NTB(M+1)
     GO TO 27
  23 LGO(K)=-LGO(K)
     GO TO 27
  24 IF(NT-45)25,954,954
  25 M=NT+1
     DO 26 N=K,NT
     LGO(M)=LGO(M-1)
     NTB(M)=NTB(M-1)
  26 M=M-1
     NTB(K)=NS
     LGO(K)=0
     NT=NT+1
  27 LM=2
     GO TO(1,31,28),NN
  28 J=J+1
     IF(KCH(J)-10)919,28,29
  29 IF(KCH(J)-46)919,1,30
  30 IF(KCH(J)-50)919,31,28
  31 GO TO(919,32),LM
  32 IP=II
     GO TO(200,200,201),IEX
 200 II=4591
     WRITE(1'II)(KODE(K),K=1,1140),(IT(N),N=1,20),(IT(M),M=22,26)
 201 DO 202 K=1,NT
     NTA(K)=NTB(K)
 202 LGA(K)=LGO(K)
     IT(2)=IP
     KODE(6)=0
     IT(6)=1
     IEX=3
     IT(3)=0
     NMB=0
     WRITE(1,901)
 901 FORMAT(/10X,4HEDIT)
     GO TO (33,34),ISW
  33 WRITE(3,100)(NTB(K),LGO(K),K=1,NT)
  34 RETURN
 961 IE=7
 954 IE=IE+35
 919 IE=19
     RETURN
     END
```

PROGRAM DLT

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE DLT
      DIMENSION IOP(50,5),IDT(90),IDS(2),IDL(2)
      DIMENSION IDD(50),ICPT(95,6)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (IOP(1,1),KODE(891)),(IDD(1),KODE(461))
      EQUIVALENCE (JS,IT(9)),(JN,IT(10))
      EQUIVALENCE (LAC,IDAT(90,1)),(LT,IDAT(90,2))
      EQUIVALENCE (IDS(1),IDAT(1,1)),(IDL(1),IDAT(1,2))
      EQUIVALENCE (NPV,KODE(6))
      EQUIVALENCE(KP,IT(7))
      GO TO (205,206),KP
  205 IM=1
      IP=1
      NN=277-IDD(1)
      DO 1 K=1,NV
    1 IOP(K,4)=NN-1
      DO 4 K=2,290
      IF(IDD(K)-268)100,4,101
  100 IF(IDD(K)-99)937,5,101
  101 M=IDD(K)/100
      IF(M)937,937,102
  102 GO TO(103,937,937,115,937),M
  103 GO TO(937,104),NN
  104 GO TO(105,106),IP
  105 IP=2
      II=1
      IC=KODE(ID+2)+ID-1
      READ(3'II)ISF,IFR,NCP,((ICPT(J,M),M=1,6),J=1,NCP)
  106 II=IC+4*(IDD(K)-101)
      DO 110 J=1,NCP
      DO 109 M=1,3
      N=II+M
      IF(KODE(N)-ICPT(J,M))110,109,110
  109 CONTINUE
      GO TO 112
  110 CONTINUE
      GO TO 4
  112 DO 113 M=1,6
  113 ICPT(J,M)=0
      GO TO 4
  115 M=IDD(K)-400
      IF(M)937,937,2
    2 IOP(M,4)=2-NN
      IM=2
    4 CONTINUE
    5 GO TO (73,6),IP
    6 J=1
      DO 66 K=1,NCP
      IF(ICPT(K,1))930,66,60
   60 IF(J-K)61,65,932
   61 DO 64 M=1,6
      ICPT(J,M)=ICPT(K,M)
   64 ICPT(K,M)=0
   65 J=J+1
   66 CONTINUE
      K=NCP
      NCP=J-1
      ISF=1
      II=1
      WRITE(3'II)ISF,IFR,NCP,((ICPT(J,M),M=1,6),J=1,K)
      IT(4)=2
      GO TO(900,74),IM
   73 GO TO (937,74),IM
  206 DO 207 K=1,NV
  207 IOP(K,4)=1
   74 IV=ID+KODE(ID)-1
      IFT=IFT+1
      CALL SHF  (IFT,90)
      DO 8 K=1,IFT
    8 IDT(K)=0
      J=1
      DO 15 K=1,NV
      IF(IOP(K,4))930,15,9
    9 LL=IV+K
      II=KODE(LL)
      IF(II)930,11,10
   10 IDT(II)=1
   11 IF(J-K)12,14,930
   12 DO 13 M=1,5
   13 IOP(J,M)=IOP(K,M)
      II=IV+J
      KODE(II)=KODE(LL)
   14 J=J+1
   15 CONTINUE
      NPV=J-1
      DO 150 M=1,NPV
  150 IOP(M,4)=0
      N=IV+J
      DO 16 K=J,40
      IOP(J,1)=0
      KODE(N)=0
   16 N=N+1
      J=1
      N=IV+1
      II=IV+NV
      DO 24 K=1,IFT
      IF(IDT(K))930,17,19
   17 IF(IDS(K))930,24,18
   18 CALL AJS  (IDS(K),IDL(K),0)
      IF(IE)900,24,900
   19 IDT(J)=0
      IF(J-K)20,23,932
   20 IDS(J)=IDS(K)
      IDL(J)=IDL(K)
      DO 21 M=N,II
      IF(KODE(M)-K)21,22,21
   21 CONTINUE
      IF(K-IFT)932,23,932
   22 KODE(M)=J
   23 J=J+1
   24 CONTINUE
      IT(4)=5
      IFT=J-1
      DO 26 K=J,90
      IDS(K)=0
```

```
 26 IDL(K)=0
 28 RETURN
937 IE=5
932 IE=IE+2
930 IE=IE+30
900 RETURN
    END
```

PROGRAM RDLL

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE RDLL
      DIMENSION LGO(1),NM(5)
      DIMENSION KTM(149),KCH(1),KWT(299),NTB(1),NVA(2),NVB(2),NVC(2)
      DIMENSION ICD(50)
      DIMENSION NVAR(50,5)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (KWT(1),KODE(451)),(KCH(1),KODE(457))
      EQUIVALENCE(LGO(1),KODE(796))
      EQUIVALENCE (IB,ICD(11))
      EQUIVALENCE (IT3,IT(3))
      EQUIVALENCE (NP,KODE(4))
      EQUIVALENCE (NPP,KODE(5))
      EQUIVALENCE (NPV,KODE(6))
      EQUIVALENCE (ICARD,KODE(7))
      EQUIVALENCE (NTB(1),KODE(751))
      EQUIVALENCE (ICD(1),KODE(841))
      EQUIVALENCE (NVAR(1,1),KODE(891))
      EQUIVALENCE (IT1,IT(1)),(IT2,IT(2)),(IT6,IT(6)),(ITC,IT(8)),
     X(KWT1,KWT(1)),(KWT2,KWT(2)),(KWT3,KWT(3)),(KWT4,KWT(4)),(KWT5,KWT
     X(5))
      EQUIVALENCE (NVA(1),NVAR(1,1)),(NVB(1),NVAR(1,4)),(NVC(1),NVAR(1,5
     1))
      NX=1
      NCX=1
      KF=(IEX-1)*1140
      GO TO (3,2),ITC
    2 IF (NAP) 1900,2000,1900
 1900 II=6591
      WRITE(1'II)(KODE(M),M=451,1140)
 2000 II=6191
      READ(1'II)(KODE(M),M=750,1140)
      ITC=1
    3 GO TO(40,40,850),ICARD
   40 IF(IE)203,202,203
  203 M=1
      IF(LGO(NMB)-210)2031,2031,2030
 2030 M=128
 2031 IT6=IT6-LGO(NMB)/M
      IE=0
      GO TO 210
   81 K=0
   82 K=K+1
      IF(KCH(K)-10)85,82,83
   83 DO 84 N=1,5
      IF(KCH(K)-NM(N))85,84,85
   84 K=K+1
      IT6=1
      NMB=0
      ICARD=2
  202 LPV=L
      NPV=NV
      NPP=NP
      NMB=NMB+1
      IF(NMB-45)210,207,954
  207 GO TO(209,208,210),IEX
  206 FORMAT(55H ONLY ONE MORE STATEMENT CAN BE ENTERED IN THIS PROGRAM)
  208 GO TO(209,210),ICARD
  209 WRITE(1,206)
  210 IF(NV-NPV)2203,2203,2200
 2200 M=NPV+1
      DO 2201 N=M,NV
      NVA(N)=0
      NVB(N)=0
 2201 NVC(N)=0
 2203 NV=NPV
      NP=NPP
      GO TO(2205,91),ICARD
 2205 DO 2204 M=1,6
 2204 KWT(M)=10
      KWT3=45
      GO TO (304,304,301),IEX
  304 NTB(NMB)=256*NMB
      GO TO 3020
  301 IF(IT3-NMB)3000,3020,930
 3000 IF(LGO(NMB))303,3002,302
  302 II=IT2
      M=LGO(NMB)
      READ(3'II)(KTM(K),K=1,M)
      IT2=IT2+M
      NX=2
      GO TO 890
  303 IT2=IT2-LGO(NMB)
 3002 IT3=NMB
 3020 NCD=NTB(NMB)/256
      NCF=NTB(NMB)-256*NCD
      GO TO 560
  850 NM(1)=24
      NM(2)=11
      NM(3)=23
      NM(4)=15
      NM(5)=10
   85 NMB=NMB+1
      IF(NMB-45)86,86,954
   86 DO 87 M=1,6
   87 KWT(M)=10
      KWT3=45
      KK=6
   51 READ(2,52)NCD,NCF,(KTM(M),M=1,74)
   52 FORMAT(I2,1X,I2,1X,74A1)
      IF(NCD)930,53,54
   53 IF(NCF)930,62,54
   62 IF(NCX-NTB(NMB))960,63,960
   54 IF(NTB(NMB))930,55,957
   55 NCX=NCD*256+NCF
      NTB(NMB)=NCX
      IF(NMB-1)930,56,550
  550 IF(NCX-NTB(NMB-1))958,959,56
   56 IF(NCD)930,580,560
  560 M=NCD/10
      IF(M)58,58,57
   57 KWT1=M
   58 KWT2=NCD-10*M
  580 IF(NCF)930,63,59
   59 M=NCF/10
      N=NCF-10*M
      IF(N)930,61,60
   60 KWT5=N
   61 KWT4=M
   63 GO TO(1060,930,630),ICARD
 1060 CALL KYBRD(KWT(1))
      KK=J
      IF(KK-299)1061,1062,1062
 1062 IF(KWT(KK-1)-45)946,1061,946
  630 J=74
      DO 64 M=1,74
      IF(KTM(J)-IB)65,64,65
   64 J=J-1
   65 MMM=J
      J=1
   11 DO 22 I=1,50
      IF(KTM(J)-ICD(I))22,12,22
   22 CONTINUE
      GO TO 953
   12 KK=KK+1
      IF(KK-297)171,171,946
  171 KWT(KK)=I-1
      IF(I-46)13,100,107
  100 IF(J-MMM)13,25,25
  107 IF(I-49)13,953,13
   13 J=J+1
      IF(J-MMM)11,11,30
   30 KK=KK+1
      KWT(KK)=51
      IF(KK-290)31,31,946
   31 DO 609 MM=1,6
      KK=KK+1
  609 KWT(KK)=10
      GO TO 51
   25 KK=KK+1
      KWT(KK)=50
 1061 KWT(KK+1)=10
      J=1
      DO 26 M=1,KK,2
      KTM(J)=KWT(M)*256+KWT(M+1)
   26 J=J+1
      M=J-1
      LGO(NMB)=M
  890 II=IT6+KF
      IT6=IT6+M
      IF(IT6-1140)891,891,910
  891 WRITE(1'II)(KTM(K),K=1,M)
      GO TO (896,92),NX
  896 GO TO (899,899,81),ICARD
   91 M=LGO(NMB)
      II=IT6+KF
      IT6=IT6+M
      READ(1'II)(KTM(K),K=1,M)
   92 J=1
      DO 93 K=1,M
      KWT(J)=KTM(K)/256
      KWT(J+1)=KTM(K)-256*KWT(J)
   93 J=J+2
      KK=J-1
```

```
899 J=299
    M=KK-6
    DO 96 K=1,M
    IF(KWT(KK)-51)95,94,95
 94 J=J+6
 95 KWT(J)=KWT(KK)
    J=J-1
 96 KK=KK-1
    J=J+1
    IF(J-81)946,946,97
 97 M=61
    DO 98 K=J,299
    KWT(M)=KWT(K)
 98 M=M+1
    KODE(932)=298-J
    I=0
    KODE(510)=0
    J=1
    RETURN
960 IE=1
959 IE=IE+1
958 IE=IE+1
957 IE=IE+3
954 IE=IE+1
953 IE=IE+7
946 IE=IE+16
930 IE=IE+20
910 IE=IE+10
    GO TO(901,901,900),ICARD
900 ICARD=2
901 RETURN
    END
```

PROGRAM SCA

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE SCA(LR)
      DIMENSION LOOKM(10)
      DIMENSION NVAR(50,5),IOP(56,3),MFL(95,6),KPG(4,10)
      DIMENSION IOOP(56),MF4(60),MF5(60),MF6(60)
      DIMENSION KCH(1),LOOK(10)
      DIMENSION KCM(1),KCP(1)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (NVAR(1,1),KODE(891))
      EQUIVALENCE (IOOP(1),KODE(451)),(MF4(1),MFL(1,4))
      EQUIVALENCE (MF5(1),MFL(1,5)),(MF6(1),MFL(1,6))
      EQUIVALENCE(IT(1),IT1),(IT(2),IT2),(NPV,KODE(6))
      EQUIVALENCE (KCH(1),KODE(511))
      EQUIVALENCE (KCP(1),KODE(512)),(KCM(1),KODE(510)),(KCH1,KCH(1))
      EQUIVALENCE (LOOK(1),KODE(737))
      EQUIVALENCE (IIZ,KODE(729)),(IM,KODE(730))
      EQUIVALENCE (LV,KODE(731)),(KV,KODE(732))
      EQUIVALENCE (KPV,KODE(733)),(KZ,KODE(734)),(K2,KODE(735))
      EQUIVALENCE (K1,KODE(736))
      EQUIVALENCE (LOOKM(1),KODE(736))
      EQUIVALENCE (ILMS,KODE(932)),(IOF,KODE(933))
      EQUIVALENCE (IOP1,IOP(1,1)),(NVAR1,NVAR(1,1)),(MFL1,MFL(1,1))
      EQUIVALENCE (IT7,IT(7)),(IT4,IT(4))
      EQUIVALENCE(IT9,IT(9)),(IT10,IT(10))
      EQUIVALENCE (NP,KODE(4))
      EQUIVALENCE (KPG(1,1),KODE(411)),(ICARD,KODE(7))
      LR=1
      IIZ=1
      IM=1
  201 I=I+1
  200 KV=KCH(I)
      IF (KV-10) 297,2001,202
 2001 IF(KCP(I)-10)297,201,2002
 2002 IF(KCP(I)-45)201,297,201
  202 IF(KV-37)205,297,297
  205 DO 206 N=1,6
  206 LOOK(N)=10
      LV=0
  207 LV=LV+1
      IF(LV-10)2070,2070,969
 2070 LOOK(LV)=KV
      I=I+1
      KV=KCH(I)
      IF(KV-37)208,209,209
  208 IF(KV-10)207,209,207
  209 DO 210 M=1,3
      N=M+M
  210 LOOK(M)=256*LOOKM(N)+LOOK(N)
      KPV=KCM(J)
      KZ=KPV-999
      IF (KZ) 2101,2101,2100
 2101 KZ=1
 2100 IF(LV-1)903,2181,2102
 2102 GO TO(2103,2104),IIZ
 2103 II=1
      READ(5'II)((IOP(M,N),N=1,3),IOOP(M),M=1,56)
      IIZ=2
 2104 CALL SERCH(IOP1,56,56)
```

```
      IF(L)930,2181,211
  211 GO TO (214,2112,919,963,212),KZ
  212 LR=3
      IT10=0
      IT9=L
      IF (L-6)9000,2121,9000
 2121 IT9=39
      GO TO 9000
 2112 IF(L-6)903,213,903
  213 IT10=-10
      LR=3
      GO TO 9000
  214 IF(L-6)215,903,2179
  215 GO TO(219,220,2190,220,217),L
  217 IF(KPV-710)904,2171,904
 2171 KCM(J)=10
 2179 KCH(J)=IOOP(L)
      IF (KCH(J)-209) 299,501,299
 5012 IF(KCH(I)-50)966,966,5010
 5011 IF(KCH(I)-10)966,5010,966
 5010 I=I+1
  501 IF(KCH(I)-49)5011,502,5012
  502 KPV=J+1
      LV=1
      J=J+2
      I=I+1
      N=0
  504 N=N+1
      K1=KCH(I)
      K2=KCP(I)
      IF(K2-49)503,506,503
  503 IF(K1-49)507,505,507
  506 K2=10
      LV=2
  507 KCH(J)=K1*256+K2
      J=J+1
      GO TO(508,509),LV
  508 I=I+2
      IF(I-ILMS)504,504,966
  505 N=N-1
      I=I-1
  509 KCH(KPV)=N
      I=I+2
      GO TO 200
  300 IF (KCH1-1003) 221,321,221
  221 M=239
      N=J
      DO 3000 I=1,J
      KCH(M)=KCH(N)
      N=N-1
 3000 M=M-1
      I=M+1
      IOF=1
      J=1
      GO TO 9000
 2190 GO TO (2191,220,220),IEX
 2191 IT7=2
      LR=7
      RETURN
  219 IEX=1
      NPV=0
  222 LR=2
      RETURN
  220 IT4=4
      GO TO 222
 2181 IF(KZ-5)2182,903,2182
 2182 CALL SERCH(NVAR1,50,NV)
      IF(L)930,302,301
  301 KCH(J)=400+L
      GO TO(299,903,919,3010),KZ
 3010 IF(LV-1)916,916,3174
  302 IF(LV-1)303,303,304
  303 GO TO(319,903,919,916),KZ
  304 GO TO (305,310,310,310),KZ
  305 IF(NP)310,310,3051
 3051 DO 307 L=1,NP
      DO 306 N=1,3
      IF(LOOK(N)-KPG(N,L))307,306,307
  306 CONTINUE
      KCH(J)=100+L
      GO TO 299
  307 CONTINUE
  310 GO TO(309,308),IM
  309 II=4
      READ(3'II)((MFL(L,M),M=1,6),L=1,NCP)
      IM=2
  308 CALL SERCH(MFL1,95,NCP)
      IF(L)930,318,311
  311 GO TO(312,314,316,3115),KZ
 3115 GO TO(915,915,3173),IEX
  312 GO TO(3122,3122,3121),IEX
 3121 IF(L-IT1)3122,319,3122
 3122 IF(NP-10)3120,944,944
 3120 NP=NP+1
      DO 313 N=1,3
  313 KPG(N,NP)=LOOK(N)
      KCH(J)=100+NP
      GO TO 299
  314 LR=3
      IT10=MF6(L)
      IT9=MF4(L)
      GO TO 315
  316 LR=LR+4
      IT1=L
  315 II=MF5(L)
      GO TO 9000
  318 GO TO(319,950,950,3173),KZ
 3173 KCH(J)=0
 3174 J=J+1
      DO 3172 N=1,3
      KCH(J)=LOOK(N)
 3172 J=J+1
      KCH(J)=10
      GO TO 299
  319 IF(NV-29)3190,940,940
 3190 NV=NV+1
```

```
      DO 320 N=1,3
 320  NVAR(NV,N)=LOOK(N)
      KCH(J)=NV+400
      GO TO 299
 297  KCH(J)=KV
      IF(KV-50)298,300,298
 298  I=I+1
 299  J=J+1
      GO TO 200
 969  IE=3
 966  IE=IE+3
 963  IE=IE+13
 950  IE=IE+6
 944  IE=IE+4
 940  IE=IE+10
 930  IE=IE+11
 919  IE=IE+1
 918  IE=IE+2
 916  IE=IE+1
 915  IE=IE+11
 904  IE=IE+1
 903  IE=IE+3
      LR=3
 321  LR=LR+3
9000  RETURN
      END
```

PROGRAM SCB

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE SCB(LR)
      DIMENSION ISK(6),ISKK(7)
      DIMENSION IPT(10)
      DIMENSION KCLA(1),KCLB(1)
      DIMENSION NTB(1)
      DIMENSION DATA(1)
      DIMENSION NVA(1),NVB(1)
      DIMENSION KSP(1),KSM(1),KSP2(1),KCP(1),KCP2(1)
      DIMENSION KCH(1), KSG(1), NVAR(50,5)
      DIMENSION LRPT(15)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (NVAR(1,1),KODE(891))
      EQUIVALENCE (KCH(1),KODE(511))
      EQUIVALENCE (KSG(1),KODE(461))
      EQUIVALENCE (LRPT(1),KODE(461))
      EQUIVALENCE(XNX,DATA(572))
      EQUIVALENCE (DATA(1),KODE(2))
      EQUIVALENCE (NTB(1),KODE(751)),(ICARD,KODE(7))
      EQUIVALENCE (IPT(1),KODE(451))
      EQUIVALENCE (IEL,IPT(1)),(IAR,IPT(2)),(IFL,IPT(3))
      EQUIVALENCE (KPN,IPT(4)),(LL,IPT(5)),(IRPT,IPT(6))
      EQUIVALENCE (IDF,IPT(7)),(ICC,IPT(8))
      EQUIVALENCE (ILX,IPT(9)),(ITHN,IPT(10))
      EQUIVALENCE (NVA(1),NVAR(1,4)),(NVB(1),NVAR(1,5))
      EQUIVALENCE (ISKK(2),ISK(1))
      EQUIVALENCE(KODE(922),ISKK(1))
      EQUIVALENCE (KSP(1),KODE(462)),(KSM(1),KODE(460)),
     1(KCP(1),KODE(512)),(KCP2(1),KODE(513)),(KSP2(1),KODE(463))
      EQUIVALENCE (KODE(930),IFX),(KODE(931),IFPC)
      EQUIVALENCE (KODE(932),LETF),(KODE(933),IOF)
      EQUIVALENCE (KCLA(1),KODE(972)),(KCLB(1),KODE(1022))
      EQUIVALENCE (FRC,KODE(936)),(M,KODE(932)),(N,KODE(984)),
     1(KV,KODE(985)),(KSMJ,KODE(986)),(KZ,KODE(987)),(NP,KODE(988)),
     2(KQ,KODE(989)),(MM,KODE(1034)),(KPV,KODE(1035)),(JSV,KODE(1036)),
     3(KS,KODE(1037)),(ND,KODE(1038))
      EQUIVALENCE (IT7,IT(7))
      IT7=1
      LR=1
      DO 19 M=1,10
      ISKK(M)=0
   19 IPT(M)=0
  100 IF(J-49-I)1001,946,946
 1001 KV=KCH(I)
      IF (KV-43) 1006,1002,1006
 1002 KSMJ=KSM(J)
      IF (KSMJ/100-1) 1004,1005,1003
 1003 IF (KSMJ-223) 1004,1005,1004
 1004 KSMJ=KSM(J-1)
      IF ((KSMJ-202)/3-1) 1006,1005,1006
 1005 IFX=1
 1006 ILX=1
      DO 601 N=1,11
      IF (KV-KCLA(N)) 603,603,601
  601 CONTINUE
      GO TO 930
  603 KQ=KCLB(N)
      IF (KQ-5) 605,605,604
  604 I=I+1
  605 GO TO (223,187,180,190,1008,136,132,800),KQ
 1008 KZ=KCP(I)+2
      DO 1009 M=1,KZ
      KSG(J)=KCH(I)
      J=J+1
 1009 I=I+1
      GO TO 100
  223 IF (KSM(J)-212) 224,1,224
    1 NP=0
      KQ=KCH(I)
      ND=0
      IF (IDF) 2,2,18
    2 NP=10*NP+KQ
    3 I=I+1
      KQ=KCH(I)
      IF (KQ-9) 2,2,4
    4 IF (KQ-45) 8,5,8
    5 I=I+1
      KQ=KCH(I)
   18 IF (KQ-9) 6,6,8
    6 ND=10*KQ
      I=I+1
      KQ=KCH(I)
      IF (KQ-9) 7,7,8
    7 ND=ND+KQ
      I=I+1
      IF (KCH(I)-9) 961,961,8
    8 IF(NP-99)10,10,961
   10 GO TO(15,17,12),IEX
   17 GO TO(15,12),ICARD
   12 NP=256*NP+ND
      DO 13 M=1,45
      IF(NP-NTB(M))13,14,13
   13 CONTINUE
      GO TO 911
   15 IF(ND)911,16,911
   16 M=NP
   14 KSG(J)=449+M
      GO TO 239
  224 XNX=0.
      IF (IDF) 225,225,228
  225 XNX=XNX*10.+KCH(I)
      I=I+1
      IF (KCH(I)-9) 225,225,226
  226 IF (KCH(I)-45)231,227,231
  227 IDF=1
      I=I+1
  228 FRC=.1
  229 IF (KCH(I)-9) 230,230,2291
 2291 IF (KCH(I)-45) 231,2292,231
 2292 IF (KCP(I)-50)917,231,917
  230 XNX=XNX+FRC*KCH(I)
      FRC=FRC/10.
      I=I+1
      GO TO 229
  231 DO 2339 M=1177,1190
      IF (XNX-DATA(M)) 2339,2333,2339
```

```
2333 KSG(J)=M-791
     GO TO 239
2339 CONTINUE
     N=152
     IF (KON) 237,237,234
 234 DO 236 M=1,KON
     IF(XNX-DATA(N))236,238,236
 236 N=N+1
 237 KON=KON+1
     IF(KON-54)2370,2370,933
2370 CALL MOV (N,572,1)
 238 KSG(J)=N+149
 239 IDF=0
     GO TO 166
 800 MM=KV-700
 801 GO TO(400,402,449,450,330,706,706,710,712,904,820,820,820),MM
 820 IF (IOF) 930,962,821
 821 IOF=0
     KSG(J)=MM+194
     GO TO 4005
 706 KSG(J)=215-2*MM
 709 KSP(J)=400
     GO TO 401
 710 KSG(J)=258
     KSP(J)=388
     GO TO 401
 712 KPV=KSM(J)
     IF (KPV-243)715,714,713
 713 IF (KPV-269)715,714,715
 714 KSG(J)=400
     GO TO 166
 715 KSG(J)=231
 166 J=J+1
     GO TO 100
 400 IAR=1
     ICC=0
     KSG(J)=222
4005 KSP(J)=265
 401 J=J+2
     GO TO 100
 402 JSV=J
     IFL=IFL+1
     J=J+1
     KSG(J)=265
     GO TO 4005
 450 KV=242
4501 IF (ISK(LL)) 4502,902,4503
4503 LL=LL-2
     KSG(J)=266
     J=J+1
     IFL=IFL-1
     GO TO 4501
4502 LL=LL+1
     ISK(LL)=1
 404 IF (IOF) 930,4062,4508
4062 KSG(J)=266
     J=J+1
     IOF=1
```

```
4508 IF (IAR) 406,406,405
 449 LL=LL+1
     ISK(LL)=-1
     IF (KSM(J)-268) 4492,4491,4492
4491 J=J-1
4492 IF ((KSG(JSV)-207)/6-1) 902,4490,902
4490 KV=241
 405 KSG(J)=266
     J=J+1
     IAR=0
 406 IF (ILX-4) 4060,4068,4060
4060 KSG(J)=266
     J=J+1
     GO TO (4061,879,4061),ILX
4061 KSG(J)=268
     J=J+1
     KSG(J)=KV
4068 IF (NV) 930,4067,4063
4063 DO 4066 N=1,NV
     IF (ILX-4) 4064,4065,930
4064 IF (NVB(N)) 4065,4066,4066
4065 NVB(N)=1
4066 CONTINUE
4067 GO TO (4005,902,878,193),ILX
 132 IF (IFL)902,902,133
 133 KSG(JSV)=KV
     KSG(J)=266
     J=J+1
     KSG(J)=262
     GO TO 4005
 136 KSG(J)=KV
     M=KV-400
     IF (M) 166,166,1779
1779 IF(NVA(M)+NVB(M))1782,1781,166
1781 IF (KSM(J)-204) 1780,1900,1780
1900 NVB(M)=1
     NVA(M)=1
     GO TO 166
1780 NVB(M)=-2
     GO TO 166
1782 NVA(M)=0
     GO TO 166
 330 IF(J-1)962,331,962
 331 IRPT=2
     II=401
     READ(5'II)(LRPT(J),J=1,15)
     J=15
     M=30
     GO TO 1900
 180 KS=KV-36
     GO TO (788,188,188,181,188,172,183,184,185,1735,870,962,962),KS
 870 IF (ISK(LL)) 874,902,872
 872 LL=LL-2
     ILX=2
     GO TO 404
 874 ILX=3
     KV=242
     LL=LL-1
```

```
      GO TO 404
  878 J=J+1
  879 IFL=IFL-1
      IF (KV-50)1187,193,1187
 1187 IF (KCP(I)-47) 1188,187,1188
 1188 KV=46
      GO TO 180
  788 IF (KCP(I)-37) 188,789,188
  789 KSG(J)=258
      I=I+1
      GO TO 88
  181 IF (KSM(J)-266) 182,188,1811
 1811 IF (KSM(J)-268) 188,182,188
  182 KSG(J)=251
      GO TO 88
  184 IF (IFX) 930,1843,1841
 1841 IFPC=IFPC-1
      IF (IFPC) 962,1842,1843
 1842 IFX=0
 1843 KPN=KPN-1
  188 KSG(J)=KV+222
   88 J=J+1
  187 I=I+1
      GO TO 100
  183 IF (IFX) 930,1832,1831
 1831 IFPC=IFPC+1
 1832 KPN=KPN+1
      GO TO 188
  185 IF (KCP(I)-9) 186,186,187
  186 IDF=1
      GO TO 187
  172 DO 178 N=1,NV
      IF(NVB(N))1721,178,178
 1721 NVA(N)=1
      GO TO 188
  178 CONTINUE
      GO TO 188
 1735 IF (IRPT-1)174,174,1736
 1736 IRPT=1
      GO TO 4490
  174 DO 175 M=1,NV
      IF(NVB(M))1744,175,175
 1744 NVB(M)=1
  175 CONTINUE
      IF (IAR) 1794,1794,176
  176 ICC=ICC+1
      IF (ICC-3)188,177,188
  177 IAR=0
      ICC=0
 1770 KSG(J)=266
      J=J+1
 1794 IF (IOF) 930,1785,188
 1785 IF (IFX) 930,1786,188
 1786 IOF=1
      GO TO 1770
  190 IF (KPN) 905,192,905
  192 ILX=4
      GO TO 404
 193 IF (IFL) 930,196,194
 194 DO 195 M=1,IFL
     KSG(J)=266
 195 J=J+1
 196 IF (IRPT) 198,198,197
 197 KSG(J)=268
     KSP(J)=212
     KSP2(J)=355
     KSP2(J+1)=266
     J=J+4
 198 KSG(J)=99
1985 IF (KSG(1)-276)202,203,205
 202 IF (KSG(1)-275)205,203,205
 203 GO TO(204,937,937),IEX
 962 IE=1
 961 IE=IE+15
 946 IE=IE+9
 937 IE=IE+4
 933 IE=IE+3
 930 IE=IE+13
 917 IE=IE+6
 911 IE=IE+6
 905 IE=IE+1
 904 IE=IE+2
 902 IE=IE+2
     LR=LR+1
 204 LR=LR+1
 205 RETURN
     END
```

PROGRAM STK

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE STK
      DIMENSION KCOP(3)
      DIMENSION NVAR(50,5)
      DIMENSION KOP(1)
      DIMENSION LGO(1)
      DIMENSION KPL(100),KSG(1),LIM(100)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (KSG(1),KODE(461))
      EQUIVALENCE (LIM(1),KODE(451))
      EQUIVALENCE (KOP(1),KODE(551)),(LG,KODE(750))
      EQUIVALENCE (LGO(1),KODE(796))
      EQUIVALENCE (NVAR(1,1),KODE(891)),(NPV,KODE(6))
      EQUIVALENCE (NUF,KODE(923)),(IFA,KODE(924)),(IPF,KODE(925))
      EQUIVALENCE (KV,KODE(926)),(NU,KODE(927))
      CALL DATSW(6,MZ)
      GO TO (5020,5030),MZ
 5020 WRITE(1,5021)(KSG(MM),MM=1,J)
 5021 FORMAT(20I4)
 5030 CONTINUE
   10 J=1
      KCOP(1)=241
      KCOP(2)=242
      KCOP(3)=264
      KT=0
      IEB=0
      L=0
      K=0
  201 MM=1
      IF (KT-268) 2002,699,2002
  699 IFM=1
  700 KQ=KPL(K)
      IF (KQ-110) 2001,2001,701
  701 IF (KQ-220) 706,706,702
  702 DO 704 N=1,3
      IF (KQ-KCOP(N)) 704,706,704
  704 CONTINUE
      IF (KQ-268) 705,706,705
  705 IF (IEB) 930,901,2001
  706 IF (IEB) 930,2001,921
 2001 GO TO (2002,2251,7561),IFM
  755 IFM=3
      KKX=LIM(L)
      IF (KKX-272) 757,700,757
 7561 IEB=IEB-1
  757 K=K+1
      KPL(K)=KKX
      L=L-1
      GO TO (2165,3033,3052,219,223,700),IXK
 2002 KT=KSG(J)
      GO TO 333
 2011 IK=JB
C   1= VAR,  2= BIN OP,  3= FUNCTION,  4= SUBSCRIPT  , 5=  (,  6=),  7= E
  202 GO TO (203,211,211,208,218,219,223,906),IK
  203 K=K+1
      KPL(K)=KSG(J)
  204 KT=KSG(J+1)
      MM=2
      GO TO 333
 2041 IK=JB
      GO TO (205,205,206,206,205,206,206,906),IK
  205 KSG(J)=259
      IK=3
      GO TO 211
  206 J=J+1
      GO TO 202
  208 K2=0
 2085 K=K+1
      KPL(K)=271+K2
      IF (KSG(J+1)-265) 2091,210,209
  209 IF (IK-4) 2091,2092,2091
 2091 L=L+1
      LIM(L)=270+K2
      J=J+1
      GO TO 201
 2092 K=K+1
      KPL(K)=KSG(J+1)
      J=J+1
      K=K+1
      KPL(K)=270+K2
      GO TO 204
  210 IF (K2) 2101,2100,2101
 2100 L=L+1
      LIM(L)=271
      GO TO 2102
 2101 LIM(L+1)=265
      LIM(L+2)=270+K2
      L=L+2
 2102 J=J+2
      GO TO 201
  211 IF(KSG(J)-209)2111,601,2110
 2110 IF (KSG(J)-268)2111,3033,2111
  601 M=KSG(J+1)+2
      DO 602 N=1,M
      K=K+1
      KPL(K)=KSG(J)
  602 J=J+1
      K=K+1
      KPL(K)=210
      GO TO 201
 2111 IF (L) 908,218,213
  213 IF (IK-2)909,218,217
  216 IXK=1
      GO TO 755
 2165 IF (L) 908,218,217
  217 CONTINUE
  788 IF (KSG(J)-269) 800,305,800
  800 IF (KSG(J)-264) 801,803,2173
  801 IF (LIM(L)-270) 2173,216,8010
 8010 IF (LIM(L)-272) 2173,802,962
  802 IF (LIM(L-1)-265)216,2173,216
  803 IF (LIM(L)-270) 804,216,962
  804 IF (LIM(L)-265) 962,805,962
  805 IF (L-1) 2181,2181,806
  806 MM=L-1
      DO 808 KK=1,MM
      NN=L-KK
      IF (LIM(NN)-265)807,808,8071
  807 IF ((LIM(NN)-239)/2-1) 962,809,962
 8072 IF(KSG(J)-268)962,2185,962
 8071 IF (LIM(NN)-272) 962,8072,962
  808 CONTINUE
  809 IF (KSG(J)-264) 2181,2181,2185
 2173 IF(KSG(J)-258)2177,2174,2177
 2174 IF(KSG(J)-LIM(L))2181,2181,2175
 2175 IF(LIM(L)-251)2176,2181,216
 2176 IF(LIM(L)-249)216,2181,216
 2177 IF(KSG(J)-LIM(L))1217,216,216
 1217 IF (LIM(L)-262)1218,1219,1218
 1218 IF (LIM(L)-260)218,1219,218
 1219 IF (KSG(J)+1-LIM(L)) 218,216,218
  218 CONTINUE
 2181 L=L+1
      LIM(L)=KSG(J)
      IF (KSG(J)-205)3011,302,3002
 3002 IF (KSG(J)-207)302,302,3001
 3001 IF (KSG(J)-222)301,302,301
  301 IF (KSG(J)-223) 3011,302,3011
 3011 IF (KSG(J)-199) 302,302,2185
  302 K2=2
      IEB=IEB+1
      GO TO 2085
  303 IF (LIM(L)-271) 3031,962,3030
 3030 IF (LIM(L-1)-265) 3032,2185,3032
 3031 IF (LIM(L)-265) 3032,3034,3032
 3032 IXK=2
      GO TO 755
 3033 IF (L) 908,2184,303
 3034 IF (L-1) 2185,2185,806
  305 IF (LIM(L)-271) 3050,2185,962
 3050 IF (LIM(L)-265) 3051,962,962
 3051 IXK=3
      GO TO 755
 3052 IF (L) 908,962,305
 2184 K=K+1
      KPL(K)=268
 2185 J=J+1
      GO TO 201
  219 IF (L) 908,908,220
  220 IF (LIM(L)-265) 222,221,2201
 2201 IF (LIM(L)-271) 222,2202,222
 2202 K=K+1
      KPL(K)=270
  221 L=L-1
      GO TO 204
  222 IXK=4
      GO TO 755
  223 IF (L) 908,225,224
  224 IXK=5
      GO TO 755
  225 IFM=2
      GO TO 700
 2251 K=K+1
      KPL(K)=99
      KV=K
      I=KV
      CALL DATSW(6,MZ)
      GO TO (559,561),MZ
  559 WRITE(1,560)(KPL(M),M=1,KV)
  560 FORMAT(20I5)
  561 CONTINUE
      GO TO 914
  333 JB=0
      IF(KT-300)1,21,21
    1 IF(KT-265)2,25,2
    2 IF(KT-266)3,26,3
    3 IF(KT-99)4,27,4
    4 IF (KT-243)5,24,5
    5 IF (KT-258)22,6,6
    6 IF(KT-300)23,28,28
   28 JB=JB+1
   27 JB=JB+1
   26 JB=JB+1
   25 JB=JB+1
   24 JB=JB+1
   23 JB=JB+1
   22 JB=JB+1
   21 JB=JB+1
      GO TO (2011,2041),MM
  914 IF(K-100)30,30,946
   30 IF(KPL(1)-263)40,40,41
   40 IF(KPL(1)-209)42,41,42
   42 IF(KPL(1)-200)41,964,964
   41 DO 31 N=1,100
   31 KOP(N)=200
      DO 36 N=1,KV
   36 LIM(N)=KPL(N)
      M=LPV+1
      IF(M-128)33,32,32
   32 IF(LG-NMB)35,35,34
   34 LG=NMB
   35 M=M-127
   33 LGO(NMB)=128*LGO(NMB)+M
      L=0
      J=1
      NUF=1
      IFA=0
      IPF=1
      KV=I
      NU=430
      RETURN
  964 IE=2
  962 IE=IE+16
  946 IE=IE+16
  930 IE=IE+9
  921 IE=IE+12
  909 IE=IE+1
  908 IE=IE+2
  906 IE=IE+5
  901 IE=IE+1
      RETURN
      END
```

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE CDR
      DIMENSION KOP(1),KPL(1),KOA(1),LGO(1)
      DIMENSION NVAR(50,5)
      DIMENSION KPP(1),KPLM1(1),KPLM2(1),KPLM3(1),KPLM4(1),KOAM(1),KOPM(
     11),NVT(1)
      DIMENSION LLSV(1)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (LLSV(1),NVAR(31,5)),(LG,KODE(750))
      EQUIVALENCE (KPL(1),KODE(451))
      EQUIVALENCE (NPV,KODE(6))
      EQUIVALENCE (NVAR(1,1),KODE(891))
      EQUIVALENCE (LGO(1),KODE(796))
      EQUIVALENCE (KOP(1),KODE(551)),(KOA(1),KODE(651))
      EQUIVALENCE (KPP(1),KODE(452)),(KPLM1(1),KODE(450)),(KPLM2(1),
     1KODE(449)),(KPLM3(1),KODE(448)),(KPLM4(1),KODE(447)),
     2(KOAM(1),KODE(650)),(KOPM(1),KODE(550)),(NVT(1),NVAR(31,4))
      EQUIVALENCE (KPL2,KODE(452))
      EQUIVALENCE (NUF,KODE(923)),(IFA,KODE(924)),(IPF,KODE(925))
      EQUIVALENCE (KV,KODE(926)),(NU,KODE(927))
      EQUIVALENCE (KODE(928),KT),(KODE(929),JX),(KODE(930),KU),
     1(KODE(931),KZ),(KODE(932),M),(KODE(933),K),(KODE(934),IK),
     2(KODE(935),MM),(KODE(936),MZ),(KODE(937),NQ),(KODE(938),JV),
     3(KODE(939),KK)
  241 L=L+1
 2415 IF (L-100) 2416,946,946
 2416 KT=KPL(J)
      JX=1
      IF (KT-270)1,23,24
    1 KU=(KT/100)+1
      GO TO (263,450,11),KU
   11 IF (KT-268) 2,274,2
    2 IF (KT-264)3,25,3
    3 IF (KT-258)4,21,21
    4 IF (KT-222)45,400,5
   45 IF (KT-209) 46,663,5
   46 IF (KT-205) 22,331,331
  663 KOP(L)=KT
      KOA(L)=KPP(J)
      KZ=KOA(L)
      J=J+1
      DO 664 M=1,KZ
      L=L+1
      J=J+1
  664 KOA(L)=KPL(J)
      J=J+1
      K=KZ+3
      GO TO 250
    5 IF (KT-223) 22,4501,22
   25 JX=JX+1
   24 JX=JX+1
   23 JX=JX+1
   22 JX=JX+1
   21 M=1
      KXX=255
      KQ=0
      GO TO (242,753,805,258,253),JX
  242 IF(J-2)962,962,2420
```

```
2420 IF (KPLM2(J)-270) 2424,2421,2424
2421 J=J-2
     GO TO 259
2424 M=2
 253 K=J-M
     M=L-1
2531 IF (KOP(M)-254)2534,2532,2534
2532 IF (KOA(M)-KPL(K))2534,2533,2534
2533 IF (KOA(M)-431)810,2538,2538
2538 GO TO (814,813),NUF
 813 NU=NU-1
     NUF=1
 814 L=L-2
     KQ=1
2536 L=L+1
 810 GO TO (2425,2537,259,910,262),JX
2534 KOP(L)=KXX
     KOA(L)=KPL(K)
     KSV=KXX
     KQ=1
     IF(KXX-255)754,2800,911
2800 IF(KOA(L)-431)2536,2801,2801
2801 KOP(L)=256
     GO TO 2536
 754 IF ((KXX-239)/2-1) 756,504,756
 756 K=1
     GO TO 489
 753 KXX=KPL(J)
     GO TO 253
2425 K=2
     KOP(L)=KPL(J)
     KOA(L)=KPLM1(J)
 520 KII=1
     IF (KOAM(L)-431) 244,243,243
 243 KII=KII+1
 244 IF (KOA(L)-431)246,245,245
 245 KII=KII+1
 246 L=L+1
     NUF=1
     IF (KII-2) 2491,249,248
 248 NU=NU-1
 249 IF (NU-430)2491,2491,2490
2490 IF (KOPM(L)-255) 2491,2492,2492
2491 NU=NU+1
     NUF=2
2492 KOP(L)=254
     KOA(L)=NU
     M=J-K
     KPL(M)=NU
 250 M=J+1
     DO 251 I=M,KV
     IK=I-K
 251 KPL(IK)=KPL(I)
     KV=KV-K
     J=J-K+1
     GO TO 241
2537 K=1
     KSV=KPL(J)
```

```
      KOP(L)=KSV
      KOA(L)=0
  489 IF (KSV-212)509,490,500
  490 CONTINUE
      KOP(L)=212
      K=KOA(L)-449
      M=0
      IF (K)4901,4901,4902
 4901 K=LPV
      M=2
      KOP(L)=255
      KOA(L)=430
      IRT=2
      GO TO 659
 4900 L=L+1
      KOP(L)=212
      GO TO 491
 4902 IF(K-NMB)4903,4903,4904
 4903 MM=0
      IF(K-LG)4907,4906,4906
 4906 MM=127
 4907 K=LGO(K)-128*(LGO(K)/128)+MM
      GO TO 4905
 4904 K=0
 4905 IF(K+IEX/2-1)911,510,491
  491 KOA(L)=244+K+M-(L+LPV)
      GO TO 510
  500 IF ((KSV-207)/6-1) 509,508,509
  504 MZ=LLSV(IFA)
      IFA=IFA-1
      IF (IFA) 912,5055,5055
 5055 KOA(MZ)=L+486-(KSV+MZ)
      KOP(L)=212
      KOA(L)=244
      IF (KSV-242) 508,5081,508
  508 IF(KV-89)5080,5080,946
 5080 IFA=IFA+1
      LLSV(IFA)=L
 5081 K=J
      GOTO 250
  509 IF (KOP(L)-204) 510,5091,520
 5091 MM=KOA(L)-400
      DO 5092 K=1,3
      L=L+1
 5092 KOA(L)=NVAR(MM,K)
      GO TO 510
  805 IF (J-5) 806,806,259
  806 IF (KPL2-271)259,807,259
  807 IF (KPP(J)-251) 259,259,258
  258 J=J+1
      GO TO 2415
  259 KOP(L)=255
      NQ=0
 2599 IF (KPLM2(J)-271) 2590,2591,2590
 2590 IF (KPLM3(J)-271) 923,2592,923
 2591 K=3+NQ
      KOA(L)=KPLM3(J)
      GO TO 781
```

```
2592 K=4+NQ
     KOA(L)=KPLM4(J)
     L=L+1
     KOA(L)=KPLM2(J)
     IF (KOA(L)-430) 781,781,780
 780 NU=NU-1
 781 L=L+1
     KOA(L)=KPLM1(J)
     IF (KOA(L)-430) 783,783,782
 782 NU=NU-1
 783 IF (NQ) 2594,7830,2594
7830 L=L+1
     NUF=1
     NU=NU+1
     GO TO 2492
2594 J=J+2
     GO TO 250
 262 JV=J-2
     IF (JV) 962,962,2620
2620 IF (KPL(JV)-270) 962,290,2621
2621 IF (KPL(JV) -430) 280,280,962
 280 KOP(L)=254
     KOA(L)=KPLM2(J)
     K=3
     GO TO 250
 510 K=2
     GO TO 250
 290 IF (KQ) 2901,2901,2902
2901 KOP(L)=255
     KOA(L)=KPLM1(J)
     L=L+1
2902 KOP(L)=254
     NQ=3
     J=J-2
     GO TO 2599
 400 JX=2
     GO TO 4501
 331 JX=4
     GO TO 4501
 450 JX=3
     KT=202
4501 DO 460 K=2,30
     M=J-K
     IF (KPL(M)-273) 460,451,459
 451 KOP(L)=KT
     GO TO (4511,4511,4510,4511),JX
4510 KOA(L)=KPL(J)
     L=L+1
4511 IK=K-2
     IF (IK) 4515,4532,4515
4515 DO 453 N=1,IK
     M=M+1
     KOA(L)=KPL(M)
 334 IF (KOA(L)-430) 453,453,452
 452 NU=NU-1
 453 L=L+1
     GO TO (4531,4531,2491,4533),JX
4531 IF (JX+1-IK) 4530,2491,4530
```

```
4530 GO TO (965,924),JX
4533 K=K+1
     L=L-1
     GO TO 250
4532 GO TO (962,962,2491,962),JX
 459 GO TO (460,460,4591,460),JX
4591 IF (KPL(M)-400)458,460,460
 458 KOP(L)=255
     KOA(L)=KPL(M)
     L=L+1
     NU=NU+1
     KOP(L)=254
     KOA(L)=NU
     KPL(M)=NU
     L=L+1
 460 CONTINUE
     GO TO 921
 263 IRT=1
 659 IF (IRT-IFA) 660,660,662
 660 DO 661 MM=IRT,IFA
     MZ=LLSV(MM)
 661 KOA(MZ)=L+244-MZ
     IFA=1
 662 GO TO (273,4900),IRT
 274 IRT=2
 273 MXT=0
     L=L-1
     K=J
 275 MXT=0
     IF (L-IPF) 287,2751,2751
2751 DO 278 KK=IPF,L
     KOK=KOA(KK)
     IF ((KOK-412)/19-1) 278,276,278
 276 KOK=KOK-430
     NVT(KOK)=1
     IF (KOK-MXT) 278,278,277
 277 MXT=KOK
 278 CONTINUE
     IF (MXT-1) 287,279,288
 279 KOP(L+1)=257
     GO TO 281
 288 L=L+1
     KOP(L)=257
     KOA(L)=MXT+430
 281 L=L+1
     KOA(L)=431
 282 IPF=L+1
     DO 284 M=1,MXT
 284 NVT(M)=1
 287 IPF=L+1
     NU=430
     GO TO (2631,250),IRT
2631 LSV=L
     L=LPV
     IF(LSV)930,380,5560
5560 DO 264 J=1,LSV
     L=L+1
     IF(L-252)264,264,909
```

```
264 KODE(L)=(KOP(J)-200)*512+KOA(J)
380 RETURN
923 IE=23
    RETURN
965 IE=3
962 IE=IE+16
946 IE=IE+22
924 IE=IE+3
921 IE=IE+9
912 IE=IE+1
911 IE=IE+1
909 IE=IE+10
910 RETURN
930 IE=30
    RETURN
    END
```

PROGRAM GETOP

```
      ENT       GETOP                                 GET00010
KODE  EQU       /7FFF                                 GET00020
KOP   EQU       /75FF                                 GET00030
KOA   EQU       /75FE                                 GET00040
KP    EQU       /75F9                                 GET00050
I     EQU       /75F5                                 GET00060
J     EQU       /75F4                                 GET00070
IE    EQU       /75F0                                 GET00080
GETOP DC        0                                     GET00090
      LD        ZRO                                   GET00100
      S     L   I                                     GET00110
      A         ONE                                   GET00120
      STO   L   2                                     GET00130
      LD    L2  KODE                                  GET00140
      SRT       9                                     GET00150
      STO   L   KOP                                   GET00160
      LD        ZRO                                   GET00170
      SLT       9                                     GET00180
      STO   L   KOA                                   GET00190
      LDX    2  43
LOOP  LD    L2  OPTB                                  GET00210
      SRT       8                                     GET00220
      S     L   KOP                                   GET00230
      BZ        MATCH                                 GET00240
      MDX    2  -1                                    GET00250
      B         LOOP                                  GET00260
      MDM   L   IE,20                                 GET00270
      B     I   GETOP                                 GET00280
MATCH LD        ZRO                                   GET00290
      SLT       4                                     GET00300
      STO   L   J                                     GET00310
      LD        ZRO                                   GET00320
      SLT       4                                     GET00330
      STO   L   KP                                    GET00340
      B     I   GETOP                                 GET00350
ZRO   DC        0                                     GET00360
ONE   DC        1                                     GET00370
OPTB  DC        0                                     GET00380
      DC        /0111   RTN                           GET00390
      DC        /0220   JMP                           GET00400
      DC        /0370   PAUSE                         GET00410
      DC        /0430   SET                           GET00420
      DC        /0540   TYPE                          GET00430
      DC        /0640   PUNCH                         GET00440
      DC        /0940   TYPEOUT                       GET00450
      DC        /0C54   GOTO                          GET00460
      DC        /0D54   LT                            GET00470
      DC        /0E54   GT                            GET00480
      DC        /0F54   EQ                            GET00490
      DC        /1054   NE                            GET00500
      DC        /1154   GE                            GET00510
      DC        /1254   LT                            GET00520
      DC        /1661   ARRAY                         GET00530
      DC        /1764   SHIFT                         GET00540
      DC        /1D62   MIN                           GET00550
      DC        /1E62   MAX                           GET00560
      DC        /1F62   INTERVALS                     GET00570
      DC        /2062   SUMF                          GET00580
      DC        /2162   LAST                          GET00590
      DC        /2263   LN                            GET00600
      DC        /2363   ATAN                          GET00610
DC       /2463    ABS                                GET00620
DC       /2563    TANH                               GET00630
DC       /2663    SUM                                GET00640
DC       /2763    MAGNITUDE                          GET00650
DC       /0780    TAB                                GET00660
DC       /2F52    SIN                                GET00670
DC       /3052    COS                                GET00680
DC       /3152    EXP                                GET00690
DC       /3252    SQRT                               GET00700
DC       /3352    NEG                                GET00710
DC       /3956     FREE
DC       /3851    LOAD/FREE
DC       /3A65    **                                 GET00740
DC       /3B53    +                                  GET00750
DC       /3C53    *                                  GET00760
DC       /3D53    /                                  GET00770
DC       /3E53    +                                  GET00780
DC       /3F53    -                                  GET00790
DC       /3655    STORE                              GET00720
DC       /3751    LOAD                               GET00730
END                                                  GET00800
```

PROGRAM RTN

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE RTN(LR)
      DIMENSION NTB(2)
      DIMENSION IAA(2),IAC(2),IDS(2),IDL(2)
      DIMENSION IACT(10,5)
      DIMENSION IBF(40)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (JS,IT(9)),(JN,IT(10))
      EQUIVALENCE (IACT(1,1),KODE(452)),(IAA(1),IACT(1,1)),
     C(IAC(1),IACT(1,2))
      EQUIVALENCE (IDS(1),IDAT(1,1)),(IDL(1),IDAT(1,2))
      EQUIVALENCE (LAC,IDAT(90,1)),(LT,IDAT(90,2))
      EQUIVALENCE (KP,IT(7))
      EQUIVALENCE (NTB(1),KODE(751)),(ICARD,KODE(7))
      LR=1
      GO TO(1,100,2),KP
  100 LE=IE
  308 IE=0
    1 IF(ID-1)939,941,101
  101 M=ID+KODE(ID)
      N=KODE(ID+3)
      IF(N)930,201,200
  200 DO 20 K=1,N
      KODE(M)=0
   20 M=M+1
  201 J=KODE(ID+1)+ID-1
      IF(M-1-J)22,90,930
   22 IFT=IFT+1
      DO 28 K=M,J
      LA=KODE(K)
      IF(LA)23,28,23
   23 IF(IFT-LA)25,25,24
   24 IFT=LA
   25 CALL AJS(IDS(LA),IDL(LA),0)
      IF(IE)209,26,209
   26 KODE(K)=0
   28 CONTINUE
  209 IFT=IFT-1
   90 LA=KODE(ID+5)
      KODE(ID+5)=0
      J=KODE(ID+6)
      KODE(ID+6)=0
      M=KODE(ID+4)
      IAC(M)=-IAC(M)
   11 ID=IAA(LA)
   12 I=ID+J
      IF(LAC)930,13,14
   13 CALL AJS(LAC,LT,1)
   14 GO TO(15,17),KP
   15 IF(IE)307,16,307
   16 RETURN
   17 IF(ID-1)939,18,1
   18 IE=LE
      GO TO 2
  941 GO TO (41,307),KP
  939 GO TO (39,307),KP
  930 GO TO (301,307),KP
   41 IE=2
  39 IE=IE+9
 301 IE=IE+30
 307 KP=2
     LE=IE
     IF (ID-1)2,2,308
   2 IF(IE-29)80,29,80
  29 WRITE(1,290)LNT
 290 FORMAT(26H  ** EXCEEDING DATA AREA -,I4,19H WORDS AVAILABLE **)
     GO TO 830
  80 II=600+IE
     READ(5'II)IB,IS
     K=IS-IB
     IF(K)830,830,81
  81 II=IB
     READ(5'II)(IBF(M),M=1,K)
     WRITE(1,82)(IBF(M),M=1,K)
  82 FORMAT(4H  **,38A2)
     IF(IE-54)830,820,830
 820 GO TO (845,821,821),IEX
 821 II=IEX*1170+1081
     READ(1'II) (KODE(K),K=1,1140),(IT(N),N=1,20),(IT(M),M=22,26)
     IT(8)=1
  30 IF(NMB-1)32,31,32
 845 NPV=0
  31 LR=LR+1
  32 LR=LR+1
     RETURN
 830 GO TO(30,84),ICARD
  84 M=NTB(NMB)/256
     N=NTB(NMB)-256*M
     AA=M+.01*N
     IF(AA)840,841,840
 840 WRITE(1,85)AA
  85 FORMAT(22H  ** STATEMENT NUMBER ,F5.2,3H **)
     GO TO 821
 841 IF(NMB-1)842,840,842
 842 M=NTB(NMB-1)/256
     N=NTB(NMB-1)-256*M
     AA=M+.01*N
     WRITE(1,843)AA
 843 FORMAT (22H  ** STATEMENT NUMBER ,F5.2,6H +1 **)
     GO TO 821
     END
```

PROGRAM JMP

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE JMP
      DIMENSION IACT(10,5),ICPT(95,6)
      DIMENSION ICB(2),ICC(2),IDL(2),IDS(2)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (IACT(1,1),KODE(452)),(JF,KODE(502)),(ICB(1),IACT(
     C1,1)),(ICC(1),IACT(1,2))
      EQUIVALENCE (IDL(1),IDAT(1,2)),(IDS(1),IDAT(1,1))
      EQUIVALENCE (KOA,IT(2))
      EQUIVALENCE (IZ,IT(8))
      IF(KOA/100-1)938,1,938
    1 IF(KOA-100)930,930,2
    2 GO TO(3,4),IZ
    3 II=6191
      WRITE(1'II)(KODE(K),K=750,1140)
      IZ=2
      IF(NAP)930,5,8
    4 IF(NAP)930,5,9
    5 IF(NCP)930,950,6
    6 DO 7 K=452,1140
    7 KODE(K)=0
      KODE(451)=1
      JF=503
      GO TO 9
    8 II=6591
      READ(1'II)(KODE(K),K=451,1140)
    9 IP=ID+KODE(ID+2)+4*KOA-405
      IF(KODE(IP+4)) 12,13,10
   10 IREF=KODE(IP+4)
      DO 11 M=3,5
      N=IP+M-2
      IF(KODE(N)-IACT(IREF,M))12,11,12
   11 CONTINUE
      GO TO 26
   12 KODE(IP+4)=0
      IREF=0
   13 IF(NAP)930,17,14
   14 DO 16 K=1,10
      DO 15 M=3,5
      N=IP+M-2
      IF(KODE(N)-IACT(K,M))16,15,16
   15 CONTINUE
      IF(ICC(K))25,25,948
   16 CONTINUE
   17 II=4
      READ(3'II)((ICPT(J,K),K=1,6),J=1,NCP)
      DO 20 K=1,NCP
      DO 18 M=1,3
      N=IP+M
      IF(KODE(N)-ICPT(K,M))950,18,20
   18 CONTINUE
      II=ICPT(K,4)
      ILP=ICPT(K,5)-II
      IF(NAP-10)19,36,36
   19 IF(JF+ILP-1141)21,21,36
   20 CONTINUE
      GO TO 950
   21 DO 22 K=1,10
```

```
      IF(ICB(K))22,23,22
   22 CONTINUE
      GO TO 930
   23 DO 24 J=1,3
      N=IP+J-2
   24 IACT(K,J)=KODE(N)
      ICB(K)=JF
      ICC(K)=-ILP
      NAP=NAP+1
      N=JF+ILP-1
      READ(3'II)(KODE(M),M=JF,N)
      KODE(JF+4)=K
      JF=JF+ILP
   25 IREF=K
      KODE(IP+4)=K
   26 IDC=ICB(IREF)
      NPAR=KODE(IDC+3)
      I=I+1
      IF(NPAR)947,260,27
  260 IF(KODE(I)/512)942,942,35
   27 J=I+NPAR-1
      DO 28 K=I,J
      IF(KODE(K)/512)943,28,943
   28 CONTINUE
      IF(KODE(J+1)/512)942,942,29
   29 IP=ID+KODE(ID)-401
      M=IDC+KODE(IDC)
      DO 34 K=I,J
      N=IP+KODE(K)
      IF(KODE(N))930,30,33
   30 IF(89-IFT)931,931,31
   31 IFT=IFT+1
      KODE(N)=IFT
      IDS(IFT)=0
      IDL(IFT)=0
   33 KODE(M)=KODE(N)
   34 M=M+1
   35 IF(ID-1)351,350,351
  350 KODE(IDC+5)=0
      GO TO 352
  351 KODE(IDC+5)=KODE(ID+4)
  352 KODE(IDC+6)=I+NPAR-1-ID
      ICC(IREF)=-ICC(IREF)
      ID=IDC
      I=ID+6
      RETURN
   36 DO 39 K=1,10
      IF(ICC(K))37,39,39
   37 NAP=NAP-1
      DO 38 M=1,5
   38 IACT(K,M)=0
   39 CONTINUE
      IF(NAP-10)40,951,951
   40 N=0
      JF=503
      IF(NAP)930,49,401
  401 IRID=KODE(ID+4)
      IRI=I-ID
      IRIP=IP-ID
   41 J=0
      DO 46 K=1,10
      IF(ICB(K))930,46,42
   42 IF(ICB(K)-JF)46,48,43
   43 IF(J)930,45,44
   44 IF(ICB(K)-ICB(J))45,930,46
   45 J=K
   46 CONTINUE
      IF(J)930,930,47
   47 K=J
      CALL MOV ((JF+1)/2,(ICB(K)+1)/2,ICC(K)/2)
      ICB(K)=JF
   48 JF=JF+ICC(K)
      N=N+1
      IF(N-NAP)41,4901,930
 4901 ID=ICB(IRID)
      I=ID+IRI
      IP=ID+IRIP
   49 IF(JF+ILP-1141)21,952,952
  952 IE=1
  951 IE=IE+1
  950 IE=IE+3
  948 IE=IE+1
  947 IE=IE+3
  943 IE=IE+1
  942 IE=IE+4
  938 IE=IE+7
  931 IE=IE+1
  930 IE=IE+30
      RETURN
      END
```

PROGRAM STV

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE STV
      DIMENSION ICB(50),IDS(2),IDL(2),KCH(299),DATA(2),IKB(4)
      DIMENSION KCP(2)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (KOP,IT(1)),(KOA,IT(2)),(JS,IT(9)),(JN,IT(10))
      EQUIVALENCE (KODE(2),DATA(1)),(XNX,DATA(572))
      EQUIVALENCE (IDS(1),IDAT(1,1)),(IDL(1),IDAT(1,2))
      EQUIVALENCE(ICB(1),KCH(90)),(ISLH,ICB(16))
      EQUIVALENCE(KCP(1),KCH(2))
      EQUIVALENCE (KCH1,KCH(1))
      NN=1
      II=501
      READ(5'II)(ICB(K),K=1,50)
      IF(KOA/100-4)938,1,938
    1 K=KODE(ID)+KOA+ID-401
      LC=KODE(K)
      IF(LC)930,2,4
    2 IF(89-IFT)931,931,3
    3 IFT=IFT+1
      LC=IFT
      KODE(K)=LC
    4 IT1=0
      JN=IDL(LC)
      CALL DATSW(15,JSW)
      ICNT=219*JSW-139
      N=31
      DO 72 M=1,3
      I=I+1
      K1=KODE(I)/256
      K2=KODE(I)-256*K1
      KCH(N-1)=ICB(K1+1)
      KCH(N)=ICB(K2+1)
   72 N=N+2
      WRITE(1,901)(KCH(M),M=30,35)
  901 FORMAT(7H ENTER ,6A1)
      GO TO (73,74),JSW
   73 II=484
      READ(5'II)(ICB(K),K=1,16)
      GO TO 75
   74 IKB(1)=40
      IKB(2)=45
      IKB(3)=46
      IKB(4)=15
   75 M=0
      GO TO 5
  500 GO TO(5,501),JSW
  501 IF(LMN-ICNT)380,5,380
  400 PAUSE
    5 LSS=M
      GO TO(60,61),JSW
   60 READ(2,7)(KCH(MJ),MJ=1,80)
    7 FORMAT(80A1)
      LMN=80
      DO 9 J=1,80
      DO 8 K=1,16
      IF(KCH(J)-ICB(K))8,80,8
    8 CONTINUE
```

```
      GO TO 380
   80 KCH(J)=K-1
      IF(K-16)9,81,380
   81 LMN=J
      IF(KCP(J)-ISLH)380,90,380
    9 CONTINUE
      GO TO 90
   61 DO 62 MJ=1,6
   62 KCH(MJ)=10
      CALL KYBRD(KCH1)
      J=J-1
      IF(KCH(J)-45)380,604,380
  604 LMN=J
      KCH(J)=15
      K=J-1
      IF(K)15,15,6040
 6040 DO 609 J=1,K
      MJ=0
      KV=KCH(J)
      IF(KV-10)609,609,605
  605 IF(KV-51)606,608,380
  606 DO 607 MJ=1,4
      IF(KV-IKB(MJ))607,608,607
  607 CONTINUE
      GO TO 380
  608 KCH(J)=MJ+10
  609 CONTINUE
   90 DO 11 J=1,LMN
      IF(KCH(J)-10)20,11,12
   11 CONTINUE
      IF(LMN-ICNT)380,5,380
   12 IF(KCH(J)-15)20,15,380
   15 GO TO(150,150,43),NN
  150 IF(IGT)17,17,16
   16 I=IGT
   17 GO TO(171,170),NN
  170 CALL AJS(IDS(LC),IDL(LC),1)
  171 CALL AJS(IDS(90),IDL(90),1)
      RETURN
  380 IF(M)18,18,381
  381 GO TO(382,39),JSW
  382 M=LSS
      JS=IDS(LC)+M
   18 WRITE(1,19)
   19 FORMAT(23H ILLEGAL INPUT, REENTER)
      GO TO (400,5),JSW
   20 GO TO(200,2100,2100),NN
  200 CALL AJS(IDS(LC),IDL(LC),JN+LNT)
      IF(IE)900,21,900
   21 JS=IDS(LC)
      JN=IDL(LC)
      NN=2
 2100 XNX=0.0
      KC=KCH(J)
      MM=1
      IF(KC-10)24,36,23
   23 KC=KC-10
      GO TO(230,26,26,380,43),KC
 230 MM=2
     K=2
     GO TO 241
  24 XNX=XNX*10.+KC
     K=1
 241 J=J+1
     KC=KCH(J)
     IF(J-LMN)240,240,379
 240 GO TO(242,242,260,27),K
 242 IF(KC-10)24,250,25
  25 IF(KC-12)250,26,250
 250 GO TO(29,380),K
  26 K=3
     GO TO 241
 260 FRC=.1
     K=4
     IF(J-2)2702,2702,2701
2701 IF(KCH(J-2)-10)27,2702,2702
2702 IF(KC-10)28,380,380
  27 IF(KC-10)28,29,29
  28 XNX=XNX+FRC*KC
     FRC=FRC/10.
     GO TO 241
  29 KC=KC-9
     GO TO(290,380,380,290,50,290),KC
 290 IT1=1
 291 K=1
     GO TO 3803
  36 J=J+1
     IF(J-LMN)2100,2100,500
 379 GO TO(3800,380),JSW
3800 IT1=1
3807 K=2
3803 GO TO(3802,3801),MM
3801 XNX=-XNX
3802 IF(M-JN)3810,41,41
3810 CALL MOV (JS,572,1)
     JS=JS+1
     M=M+1
     NN=3
     GO TO(3804,5),K
3804 GO TO(36,930,930,36,930,43),KC
  39 K=JS-1
     GO TO(403,401),IT1
 403 WRITE(1,40)DATA(K)
  40 FORMAT(23H REENTER NUMBERS AFTER ,F12.4)
     GO TO 5
 401 WRITE(1,402) DATA(K)
 402 FORMAT(22H REENTER NUMBERS AFTER,E15.6)
     GO TO 5
  41 CALL AJS  (IDS(LC),IDL(LC),0)
     NN=1
     IF(IE)900,929,900
  43 JN=JN-M
     CALL FRE
     IDL(LC)=M
     RETURN
  50 K=1
```

```
      IMF=0
      KC=1
      J=J+1
      IF(J-LMN)52,52,380
   52 IF(KCH(J)-10)58,54,56
   56 IF(KCH(J)-11)380,53,380
   53 K=2
   54 J=J+1
      IF(J-LMN)55,55,380
   55 IF(KCH(J)-10)58,380,380
   57 KC=2
   58 IMF=IMF*10+KCH(J)
      J=J+1
      IF(J-LMN)59,59,64
   59 IF(KCH(J)-10)68,64,63
   68 GO TO(57,380),KC
   63 KC=KCH(J)-10
      GO TO(380,380,64,380,64),KC
   64 FRC=IMF
      GO TO(66,65),K
   65 FRC=-FRC
   66 XNX=XNX*10.**FRC
      IT1=2
      KC=KCH(J)-9
      IF(J-LMN)291,291,67
   67 GO TO(3807,380),JSW
  938 IE=7
  931 IE=IE+1
  930 IE=IE+1
  929 IE=IE+29
  900 GO TO(71,70,70),NN
   70 N=IE
      IE=0
      CALL AJS (IDS(LC),IDL(LC),0)
      IE=N
   71 RETURN
      END
```

PROGRAM WRT

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE WRT
      DIMENSION DATA(2)
      DIMENSION BF(240)
      DIMENSION IBF(80),ICR(50)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (KOP,IT(1)),(KOA,IT(2))
      EQUIVALENCE (LAC,IDAT(90,1)),(LT,IDAT(90,2))
      EQUIVALENCE (KODE(2),DATA(1))
      EQUIVALENCE (IBF(2),BF(1))
      CALL DATSW(0,JSW)
      J=5-2*JSW
      CALL DATSW(1,IFC)
      LD=7-KOP
      IF(KOP-9)1000,400,1000
 1000 CALL DATSW(13,ISW)
      GO TO(121,418),ISW
  418 IF(KOA)938,100,102
  100 IF(LAC)930,930,101
  101 LB=90
      GO TO 110
 1010 KOA=KODE(I)
  102 LB=KOA/100
      GO TO(938,938,104,108),LB
  104 IF(KOA-386)106,105,105
  105 LB=KOA+791
      GO TO 107
  106 LB=(KODE(ID+1)+ID+2*KOA-601)/2
  107 LC=1
      GO TO 111
  108 LB=ID+KODE(ID)+KOA-401
      LB=KODE(LB)
      IF(LB)930,922,109
  109 IF(IDAT(LB,1))930,922,110
  110 LC=IDAT(LB,2)
      LB=IDAT(LB,1)
  111 GO TO (122,112),IFC
  112 N=240
 1120 IF(LC-N)113,114,114
  113 N=LC
  114 DO 117 M=1,N
      IF(DATA(LB))115,116,116
  115 BF(M)=DATA(LB)-.00005
      GO TO 117
  116 BF(M)=DATA(LB)+.00005
  117 LB=LB+1
      M=0
 1170 MM=M+1
      M=M+2*LD+4
      IF(M-N)1172,1172,1171
 1171 M=N
 1172 GO TO(1180,118),LD
  118 WRITE(J,903)(BF(K),K=MM,M)
  903 FORMAT(8(1H ,F12.4))
      GO TO 1182
 1180 WRITE(J,904)(BF(K),K=MM,M)
  904 FORMAT(6(1H ,F12.4))
      WRITE(2,904)(BF(K),K=MM,M)
 1182 CALL DATSW(13,ISW)
      GO TO(121,1183),ISW
 1183 IF(M-N)1170,120,120
  120 LC=LC-N
      IF(LC)121,121,1200
 1200 GO TO(1120,1201),LD
 1201 CALL DATSW(13,ISW)
      GO TO(121,1120),ISW
  122 N=2*LD+3
 1223 IF(LC-N)1224,1225,1225
 1224 N=LC
 1225 M=LB+N-1
 1226 WRITE(J,905)(DATA(K),K=LB,M)
  905 FORMAT(1H ,7E15.6)
      GO TO (1228,1229),LD
 1228 WRITE (2,905) (DATA(K),K=LB,M)
 1229 CALL DATSW(13,ISW)
      GO TO(121,1227),ISW
 1227 LC=LC-N
      LB=M+1
      IF(LC)121,121,1223
  121 IF(ID-1)930,124,125
  124 IF(L-I)930,126,125
  125 IF(KODE(I+1)/512)126,127,126
  126 RETURN
  127 I=I+1
      GO TO(121,1010),ISW
  400 IF(KOA)955,955,401
  401 CALL DATSW(13,K)
      GO TO(4110,4010),K
 4010 II=501
      READ(5'II)(ICR(K),K=1,50)
      N=0
      DO 410 K=1,KOA
  402 I=I+1
      LC=1
      M=KODE(I)/256
      IF(M-50)403,966,406
  403 N=N+1
      IBF(N)=ICR(M+1)
  404 LC=2
      M=KODE(I)-256*M
      IF(M-50)409,966,406
  406 IF(N)930,4060,407
 4060 WRITE(J,4061)
 4061 FORMAT(1H )
      GO TO 408
  407 WRITE(J,908)(IBF(LD),LD=1,N)
  908 FORMAT(80A1)
      CALL DATSW(13,LD)
      GO TO(4071,4070),LD
 4070 N=0
  408 GO TO(404,410),LC
  409 N=N+1
      IBF(N)=ICR(M+1)
  410 CONTINUE
      IF(N)930,411,412
  412 WRITE(J,908)(IBF(LD),LD=1,N)
```

```
      RETURN
4110  I=I+KOA
 411  RETURN
4071  I=I+KOA-K
      RETURN
 966  IE=11
 955  IE=IE+17
 938  IE=IE+8
 930  IE=IE+8
 922  IE=IE+22
      RETURN
      END
```

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE LSG(KO)
      DIMENSION DATA(2),IDS(2),IDL(2),IAR(4)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (KOP,IT(1)),(KOA,IT(2)),(KP,IT(7))
      EQUIVALENCE (LAC,IDAT(90,1)),(LT,IDAT(90,2))
      EQUIVALENCE (KODE(2),DATA(1))
      EQUIVALENCE (IA,KODE(1143)),(IB,KODE(1144))
      EQUIVALENCE (IG,IAR(1)),(IH,IAR(2)),(IC,IAR(3)),(IK,IAR(4))
      EQUIVALENCE (IDS(1),IDAT(1,1)),(IDL(1),IDAT(1,2))
      EQUIVALENCE (NPV,KODE(6))
C  KP VALUES
C  1  LOAD,LOAD/FREE TEMPORARY
C  2  SIN, COS, EXP, SQRT, NEG
C  3  *,/,+,-,+
C  4  GO TO, LT, GT, EQ, NE, LE, GE - INITIALLY
C     STORE - AFTER OPERAND CLASSIFIED
C  5  STORE - INITIALLY
C  6  FREE TEMPORARY
      KO=1
      IC=0
      LL=1
      GO TO(40,1,40,120,40,40),KP
    1 IF(KOA)938,2,40
    2 IF(LAC)930,930,3
    3 LD=90
      LA=1
      GO TO 813
  167 GO TO (168,169,930),LL
  168 GO TO(923,1680),LA
 1680 IG=LD
      IH=LC
  169 LL=LL+1
      I=I+1
      KOA=KODE(I)
C    LOCATE OPERAND
   40 LA=KOA/100-2
      IF(LA)938,938,4
    4 GO TO (5,6), LA
    5 IF(KP-5)501,928,928
  501 IF(KOA-386)503,502,502
  502 LB=KOA+791
      GO TO 504
  503 LB=(KODE(ID+1)+ID+2*KOA-601)/2
  504 LC=1
      GO TO 13
    6 IF(KOA-400)938,602,608
  602 GO TO(603,938,938,604),KP
  603 IAR(LL+1)=IH-1
      GO TO 1631
  604 GO TO(938,605,605),LL
  605 IF(IH)930,907,606
  606 IF(IDS(IH))930,907,607
  607 IAR(LL+1)=IDL(IH)-1
      GO TO 1631
  608 LB=KODE(ID)+ID+KOA-401
      LD=KODE(LB)
      IF(KP-5)601,1630,6080
  601 IF(LD)930,922,7
    7 IF(IDS(LD))930,922,813
  813 LB=IDS(LD)
      LC=IDL(LD)
      IF(KOP-56)13,1610,13
 1610 CALL AJS(LAC,LT,0)
      CALL SHF(90,LD)
      IF(IE)45,1611,45
 1611 RETURN
   13 GO TO(161,16,50,161),KP
C    MULTIPLE INSTRUCTION LOAD OR STORE
  161 GO TO (164,162,162),LL
  162 IF(LC-1)930,163,923
  163 IAR(LL+1)=DATA(LB)+.5
      IF(IAR(LL+1))923,1631,1631
 1631 GO TO(938,164,170),LL
 1630 KP=4
      LC=LD
 6080 LD=LB
  164 IF(ID-1)930,165,166
  165 IF(L-I)930,170,166
  166 IF(KODE(I+1)/512)170,167,170
  170 GO TO(171,920,920,72,920,6085),KP
 6085 GO TO (6086,6082),LL
 6082 LD=IG
 6086 DO 6084 K=LB,LD
      LC=KODE(K)
      IF (LC) 930,6084,6083
 6083 CALL AJS(IDS(LC),IDL(LC),0)
      IF (IE) 45,6084,45
 6084 CONTINUE
      RETURN
  171 GO TO (16,172,172),LL
  172 LD=IG
      IF(IAR(LL+1)-IH)173,173,923
  173 GO TO(930,176,174),LL
  174 LC=IK-IC+1
      IF(LC)930,923,16
  176 LC=1
C     LOAD
   16 CALL AJS  (LAC,LT,LC)
      IF(IE)45,160,45
  160 GO TO(1601,1602,1602),LL
 1601 GO TO(1603,1602),LA
 1602 LB=IDS(LD)
 1603 GO TO(17,140,58),KP
   17 LB=LB+IC
      CALL MOV (LAC,LB,LC)
      RETURN
C     *,/,+,-,+
   50 M=0
      IF(KOP-63)51,710,51
   51 LE=KOP-58
      IF(LT-LC) 55,54,55
   54 IF(LC-1)926,61,60
   55 IF(LC-1)926,61,56
   56 IF(LT-1)927,57,927
   57 CALL MOV (572,LAC,1)
```

```
      GO TO 16
   58 N=LAC+LT-1
      DO 59 K=LAC,N
   59 CALL MOV (K,572,1)
   60 M=1
   61 N=LAC+LT-1
      GO TO(64,66,68,70),LE
   64 DO 65 K=LAC,N
      DATA(K)=DATA(K)*DATA(LB)
   65 LB=LB+M
      RETURN
   66 DO 67 K=LAC,N
      DATA(K)=DATA(K)/DATA(LB)
   67 LB=LB+M
      RETURN
   68 DO 69 K=LAC,N
      DATA(K)=DATA(K)+DATA(LB)
   69 LB=LB+M
      RETURN
   70 DO 71 K=LAC,N
      DATA(K)=DATA(K)-DATA(LB)
   71 LB=LB+M
      RETURN
C     +
  710 IA=0
      IB=0
      CALL AJS  (IA,IB,LT+LC)
      IF(IE)45,711,45
  711 CALL MOV (IA,LAC,LT)
      GO TO(714,713),LA
  713 LB=IDS(LD)
  714 CALL MOV (IA+LT,LB,LC)
      CALL AJS  (LAC,LT,0)
      IF(IE)45,712,45
  712 LAC=IA
      LT=IB
      RETURN
C    STORE
   72 GO TO (74,73,73),LL
   73 LC=IH
      LB=IG
   74 N=1
      IF(LC)930,75,77
   77 IF(IDS(LC))930,78,79
   75 IF(89-IFT)931,931,76
   76 IFT=IFT+1
      KODE(LB)=IFT
      LC=IFT
   78 N=2
   79 GO TO(80,86,88),LL
   80 IF(ID-1)939,81,82
   81 IF(L-I)939,87,82
   82 LD=KODE(I+1)/512
      IF(LD-55)83,87,89
   83 IF(LD-54)830,89,830
  830 IF(LD-40)831,87,84
  831 IF(LD-23)832,87,84
  832 IF(LD-22)833,87,89
  833 IF(LD-12)834,89,89
  834 IF(LD-4)835,87,84
  835 IF(LD-2)89,87,87
   84 IF(KODE(I+1)-512*LD)89,89,87
   86 IF(LT-1)926,880,923
   87 CALL SHF  (90,LC)
      RETURN
C STORE BY SUBSCRIPT
   88 IF(IK-IC)923,885,885
  885 IF(LT-IK+IC-1)8800,880,927
 8800 IF(LT-1)927,8801,927
 8801 CALL MOV (572,LAC,1)
      CALL AJS  (LAC,LT,IK-IC+1)
      IF(IE)45,8802,45
 8802 M=LAC+LT-1
      DO 8803 K=LAC,M
 8803 CALL MOV (K,572,1)
  880 IF(IDL(LC)-IAR(LL+1)-1)881,90,90
  881 IA=0
      IB=0
      CALL AJS(IA,IB,IAR(LL+1)+1)
      IF(IE)45,882,45
  882 GO TO(883,884),N
  883 CALL MOV (IA,IDS(LC),IDL(LC))
      CALL AJS  (IDS(LC),IDL(LC),0)
      IF(IE)45,884,45
  884 IDS(LC)=IA
      IDL(LC)=IB
      GO TO 90
   89 IC=0
      CALL AJS  (IDS(LC),IDL(LC),LT)
      IF(IE)45,90,45
   90 CALL MOV (IDS(LC)+IC,LAC,LT)
      RETURN
C    GO TO, LT, GT, EQ, NE, LE, GE
  120 KOA=KOA-244
      IF(KOP-12)920,127,1201
 1201 KOP=KOP-12
      IF(LT-1)925,1202,925
 1202 A=1.0
      GO TO (121,122,123,124,125,126),KOP
  121 A=-A
  122 IF(A*DATA(LAC))128,128,129
  123 IF (DATA(LAC)) 128,129,128
  124 IF (DATA(LAC)) 129,128,129
  125 A=-A
  126 IF(A*DATA(LAC))129,129,128
  127 CALL DATSW(14,K)
      GO TO (44,1270),K
   44 NPV=0
      KO=3
      RETURN
 1270 IF(KOA)1271,129,1271
 1271 IGT=I
      KO=2
  128 I=I+KOA-1
  129 RETURN
C    SIN, COS, EXP, SQRT, NEG
```

```
140 KOP=KOP-46
    N=LAC+LT-1
    GO TO (141,143,145,147,149),KOP
141 DO 142 K=LAC,N
    DATA(K)=SIN(DATA(LB))
142 LB=LB+1
    RETURN
143 DO 144 K=LAC,N
    DATA(K)=COS(DATA(LB))
144 LB=LB+1
    RETURN
145 DO 146 K=LAC,N
    DATA(K)=EXP(DATA(LB))
146 LB=LB+1
    RETURN
147 DO 148 K=LAC,N
    DATA(K)=SQRT(DATA(LB))
148 LB=LB+1
    RETURN
149 DO 150 K=LAC,N
    DATA(K)=-DATA(LB)
150 LB=LB+1
    RETURN
939 IE=1
938 IE=IE+7
931 IE=IE+1
930 IE=IE+2
928 IE=IE+1
927 IE=IE+1
926 IE=IE+1
925 IE=IE+2
923 IE=IE+1
922 IE=IE+2
920 IE=IE+13
907 IE=IE+7
 45 KO=4
    RETURN
    END
```

PROGRAM TRG

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE TRG
      DIMENSION DATA(2),IDS(2),IDL(2)
      DIMENSION IAR(2),AR(3)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (KOP,IT(1)),(KOA,IT(2)),(KP,IT(7))
      EQUIVALENCE (LAC,IDAT(90,1)),(LT,IDAT(90,2))
      EQUIVALENCE (KODE(2),DATA(1)),(IA,KODE(1143)),(IB,KODE(1144))
      EQUIVALENCE (IDS(1),IDAT(1,1)),(IDL(1),IDAT(1,2))
      EQUIVALENCE (IG,IAR(1)),(IH,IAR(2)),(IC,IAR(3)),(IK,IAR(4))
      EQUIVALENCE (JS,IT(9)),(JN,IT(10))
      EQUIVALENCE (IAR(2),AR(1)),(A,AR(1)),(AB,AR(2)),(AC,AR(3))
C KP VALUES
C 1  ARRAY
C 2  MIN,MAX,INT,SUMF,LAST
C 3  LN,ARCTAN,ABS,TANH,SUM,MAGNITUDE
C 4  SHIFT
C 5  **
      LL=1
      CALL DATSW(3,IDD)
 1000 IF(KOA)938,1,3
    1 IF(LAC)930,930,2
    2 LD=90
      LA=1
      GO TO(938,12,12,938,938),KP
    3 LA=KOA/100-2
      IF(LA) 938,938,4
    4 GO TO (5,10),LA
    5 GO TO(6,6,6,501,6),KP
  501 GO TO(6,945),LL
    6 IF(KOA-386)8,7,7
    7 M=KOA+791
      GO TO 9
    8 M=(ID+KODE(ID+1)+2*KOA-601)/2
    9 LC=1
      GO TO 13
   10 LB=ID+KODE(ID)+KOA-401
      LD=KODE(LB)
      IF(LD) 930,922,11
   11 IF(IDS(LD))930,922,12
   12 M=IDS(LD)
      LC=IDL(LD)
   13 GO TO(200,29,130,1300,70),KP
 1300 GO TO(45,130),LL
  130 CALL AJS  (LAC,LT,LC)
      IF(IE)900,131,900
  131 GO TO(1310,1301),LA
 1301 M=IDS(LD)
 1310 KP=KP-2
      GO TO(1311,51),KP
 1311 K=KOP-33
      N=LAC+LT-1
      GO TO(132,14,16,18,20,210),K
C LN
  132 DO 133 K=LAC,N
      DATA(K)=ALOG(DATA(M))
  133 M=M+1
      RETURN
```

```
C ARCTAN
   14 DO 15 K=LAC,N
      DATA(K)=ATAN(DATA(M))
   15 M=M+1
      RETURN
C ABS
   16 DO 17 K=LAC,N
      DATA(K)=ABS(DATA(M))
   17 M=M+1
      RETURN
C TANH
   18 DO 19 K=LAC,N
      DATA(K)=TANH(DATA(M))
   19 M=M+1
      RETURN
C SUM
   20 CALL MOV (LAC,M,1)
      LL=LAC+1
      M=M+1
      DO 21 K=LL,N
      DATA(K)=DATA(K-1)+DATA(M)
   21 M=M+1
      RETURN
C MAGNITUDE
  210 DO 211 K=LAC,N
      A=0.4342945*ALOG(DATA(M))
      IF(A)212,213,213
  212 LL=A-1
      GO TO 214
  213 LL=A
  214 A=LL
      DATA(K)=10.0**A
  211 M=M+1
      RETURN
   29 K=KOP-28
      GO TO(30,31,36,38,36),K
C MIN
   30 A=-1.0
      GO TO 32
C MAX
   31 A=1.0
   32 CALL MOV (572,M,1)
      N=M+LC-1
      DO 34 K=M,N
       IF(A*(DATA(K)-DATA(572)))34,34,33
   33 CALL MOV (572,K,1)
   34 CONTINUE
      CALL AJS  (LAC,LT,1)
      IF(IE) 900,35,900
   35 CALL MOV (LAC,572,1)
      RETURN
C INTERVALS,LAST
   36 CALL AJS  (LAC,LT,1)
      IF (IE) 900,37,900
   37 DATA(LAC)=LC-1
      RETURN
C SUMF
   38 CALL MOV (572,M,1)
```

```
      IF(LC-1)930,41,39
   39 N=M+1
      LL=M+LC-1
      DO 40 K=N,LL
   40 DATA(572)=DATA(572)+DATA(K)
   41 CALL AJS  (LAC,LT,1)
      IF(IE)900,42,900
   42 CALL MOV (LAC,572,1)
      RETURN
C SHIFT
   45 IF(LC-1)930,46,965
   46 IF(DATA(M))47,48,48
   47 NM=DATA(M)-0.5
      GO TO 49
   48 NM=DATA(M)+0.5
   49 LL=2
      I=I+1
      KOP=KODE(I)/512
      KOA=KODE(I)-512*KOP
      GO TO (50,500),IDD
   50 WRITE(3,901)I,KOP,KOA
  500 IF(KOP)930,3,943
  901 FORMAT(3I5)
   51 N=1
      IF(NM)52,66,53
   52 NM=-NM
      N=2
   53 IF(NM-LC)55,66,54
   54 NM=NM-LC
      GO TO 53
   55 IF(NM)930,66,56
   56 GO TO(57,58),N
   57 IF(NM-LC/2)60,59,59
   58 IF(NM-LC/2)60,60,59
   59 NM=LC-NM
      N=3-N
   60 IA=0
      IB=0
      CALL AJS  (IA,IB,NM)
      IF(IE)900,61,900
   61 M=IDS(LD)
      GO TO(62,64),N
   62 CALL MOV (IA,M+LC-NM,NM)
      LL=M+LC-NM-1
      N=LAC+LT-1
      DO 63 K=2,LT
      CALL MOV (N,LL,1)
      N=N-1
   63 LL=LL-1
      CALL MOV (LAC,IA,NM)
      GO TO 65
   64 CALL MOV (IA,M,NM)
      CALL MOV (LAC,M+NM,LC-NM)
      CALL MOV (LAC+LT-NM,IA,NM)
   65 CALL AJS  (IA,IB,0)
      RETURN
   66 CALL MOV (LAC,M,LC)
      RETURN
```

```
C ARRAY
  200 IF(LC-1)924,201,924
  201 IAR(2*LL-1)=KODE(2*M-1)
      IAR(2*LL)=KODE(2*M)
      GO TO(202,202,205),LL
  202 LL=LL+1
      I=I+1
      KOP=KODE(I)/512
      KOA=KODE(I)-512*KOP
      GO TO(203,204),IDD
  203 WRITE(3,901)I,KOP,KOA
  204 IF(KOP)924,2040,924
 2040 IF(KODE(I)-KODE(I-1))3,201,3
  205 IF(AC)924,2050,2050
 2050 LC=AC+1.5
      CALL AJS  (LAC,LT,LC)
      IF(IE)900,206,900
  206 AC=(AB-A)/(LT-1)
      AB=0.0
      N=LAC+LT-1
      DO 207 K=LAC,N
      DATA(K)=AB*AC+A
  207 AB=AB+1.0
      RETURN
C **
   70 LB=M
      M=0
      IF(LT-LC)75,74,75
   74 IF(LC-1)926,81,80
   75 IF(LC-1)926,81,76
   76 IF(LT-1)927,77,927
   77 CALL MOV(572,LAC,1)
      CALL AJS(LAC,LT,LC)
      IF(IE)900,781,900
  781 GO TO(78,782),LA
  782 LB=IDS(LD)
   78 N=LAC+LT-1
      DO 79 K=LAC,N
   79 CALL MOV(K,572,1)
   80 M=1
   81 N=LAC+LT-1
      DO 83 K=LAC,N
      IF(DATA(K))620,625,625
  620 IF(DATA(LB))621,622,622
  621 IA=DATA(LB)-.5
      GO TO 623
  622 IA=DATA(LB)+.5
  623 IF(IA-DATA(LB))625,624,625
  624 DATA(K)=DATA(K)**IA
      GO TO 83
  625 DATA(K)=DATA(K)**DATA(LB)
   83 LB=LB+M
      RETURN
  965 IE=20
  945 IE=IE+2
  943 IE=IE+5
  938 IE=IE+8
  930 IE=IE+3
```

```
927 IE=IE+1
926 IE=IE+2
924 IE=IE+2
922 IE=IE+2
920 IE=IE+20
900 RETURN
    END
```

PROGRAM TAB

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE TAB
      DIMENSION DATA(2)
      DIMENSION IBF(105)
      DIMENSION IDS(2),IDL(2)
      DIMENSION IRN(29),IZ(11),ICR(11),ICB(2),IVL(8),IVS(8)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (KOP,IT(1)),(KOA,IT(2)),(IRN(5),IZ(1))
      EQUIVALENCE (IV,IVL(1))
      EQUIVALENCE (LAC,IDAT(90,1)),(LT,IDAT(90,2))
      EQUIVALENCE (KODE(2),DATA(1))
      EQUIVALENCE (ICARD,KODE(7))
      EQUIVALENCE (IDS(1),IDAT(1,1)),(IDL(1),IDAT(1,2))
      EQUIVALENCE (IRN(19),ICR(1)),(IRN(17),ICB(1)),(XNX,DATA(572)),
     X(IBL,IRN(29))
      CALL DATSW(13,ISW)
      GO TO(121,418),ISW
  418 CALL DATSW(0,JSW)
      J=5-2*JSW
      CALL DATSW(1,IFC)
      LM=1
      II=0
      LMX=6+IFC
 1000 IF(KOA)938,100,102
  100 IF(LAC)930,930,101
  101 LB=90
      GO TO 110
 1182 IF(LM)930,1183,1100
 1183 WRITE(J,1184)
 1184 FORMAT(1H )
 1100 LM=LM+1
 1010 KOA=KODE(I)
  102 LB=KOA/100
      GO TO(938,938,104,108),LB
  104 IF(KOA-386)106,105,105
  105 IVS(LM)=KOA+791
      GO TO 107
  106 IVS(LM)=(KODE(ID+1)+ID+2*KOA-601)/2
  107 IVL(LM)=1
      GO TO 111
  108 LB=ID+KODE(ID)+KOA-401
      LB=KODE(LB)
      IF(LB)930,922,109
  109 IF(IDS(LB))930,922,110
  110 IVL(LM)=IDL(LB)
      IVS(LM)=IDS(LB)
  111 IF(LM-LMX)121,301,930
  121 IF(ID-1)930,124,125
  124 IF(L-I)938,1280,125
  125 IF(KODE(I+1)/512)1280,127,1280
 1280 GO TO(900,128),ISW
  127 I=I+1
      GO TO(121,1182),ISW
  128 IF(LM)900,900,301
  301 IF(II)35,302,35
  302 IRN(1)=1
      IRN(2)=10
      IRN(3)=100
      IRN(4)=1000
      DO 303 M=5,16
  303 IRN(M)=0
      IRN(6)=19264
      IRN(13)=-15040
      IRN(14)=16448
      IRN(17)=16448
      IRN(18)=24640
      IRN(19)=-4032
      IRN(20)=-3776
      IRN(21)=-3520
      IRN(22)=-3264
      IRN(23)=-3008
      IRN(24)=-2752
      IRN(25)=-2496
      IRN(26)=-2240
      IRN(27)=-1984
      IRN(28)=-1728
      IRN(29)=16448
      II=1
   35 INR=IV
      IF(LM-1)930,39,36
   36 DO 38 K=2,LM
      IF(IVL(K)-INR)38,38,37
   37 INR=IVL(K)
   38 CONTINUE
   39 DO 328 K=1,INR
      DO 40 M=1,105
   40 IBF(M)=IBL
      IST=-10-IFC
      DO 327 MN=1,LM
      IST=IST+17-2*IFC
      IF(K-IVL(MN))41,41,327
   41 NN=IST+11
      N=IST+6*IFC-6
      MM=1
      DO 3021 M=N,NN
      IBF(M)=IZ(MM)
 3021 MM=MM+1
      NS=1
      CALL MOV  (572,IVS(MN)+K-1,1)
      IF(XNX)304,327,305
  304 XNX=-XNX
      NS=2
  305 GO TO(3051,3050),IFC
 3050 XNX=XNX+.00005
 3051 DG=0.4342945*ALOG(XNX)
      LL=DG
      IS=1
      IF(DG)3052,3053,3053
 3052 IS=2
 3053 GO TO(3060,3074),IFC
 3060 M=IST+2
      IBF(IST-1)=ICB(NS)
      N=IS
      GO TO(3063,3064),IS
 3063 NP=LL+1
      GO TO 3065
```

```
3064 IF (LL) 3101,3102,3101
3101 IBF(IST+9)=24640
3102 NP=-LL
3065 MM=NP
     IF(MM-10)3068,3067,3067
3067 NL=MM/10
     IBF(IST+10)=NL
     MM=MM-10*NL
3068 IBF(IST+11)=MM
     GO TO(3069,3073,3198),N
3070 XNX=XNX/10000.
     LL=LL-4
3069 IF(LL-3)310,310,3070
3072 XNX=XNX*10000.
     LL=LL+4
3073 IF(LL+3)3072,315,315
3074 IF(LL+NS-8)307,324,324
 307 M=IST+6-LL
     GO TO(308,314),IS
 308 IBF(M-1)=ICB(NS)
     GO TO 3069
 310 IG=IRN(LL+1)
     GO TO 3170
 314 M=M+2
     IBF(IST+5)=ICB(NS)
     IF(LL+4)327,327,315
 315 XNX=XNX*10000.
     IG=IRN(LL+4)
3170 NM=1
 318 IDP=XNX
     XNX=XNX-IDP
     IFF=IG
     NN=8192
     DG=.00197601
     DO 3183 N=1,14
     IF(IDP-NN)3182,3184,3184
3182 DG=DG/2.
3183 NN=NN/2
     GO TO 319
3184 XNX=XNX+DG
 319 NN=IDP/IFF
     ISK=0
     IF(NN-10)3199,3190,3190
3190 ISK=10
     KNK=M
     IF(IFF-IG)3002,3000,324
3000 GO TO(3001,3002),NM
3001 KNK=KNK-1
     GO TO (3012,3010),IFC
3008 IBF(KNK)=0
3002 KNK=KNK-1
     IF(KNK-IST-6*IFC+5)3005,3003,3005
3003 GO TO (3012,3004),IFC
3004 KNK=KNK-1
3005 IF((IBF(KNK)+9)/9-1)3007,3006,3007
3007 IF(IBF(KNK)-9)3009,3008,3009
3009 GO TO (324,3010),IFC
3010 IF(KNK-IST-NS+1)324,3011,3011
```

```
 3011 IBF(KNK-1)=IBF(KNK)
      IBF(KNK)=1
      GO TO 3199
 3012 KNK=KNK+1
      IBF(KNK)=1
      N=3
      NP=NP+1
      GO TO 3065
 3198 M=M+1
      IF(M-IST-12)3199,327,327
 3006 IBF(KNK)=IBF(KNK)+1
 3199 IBF(M)=NN-ISK
      IDP=IDP-NN*IFF
      IFF=IFF/10
  320 M=M+1
      GO TO(3200,3201),IFC
 3200 IF(M-IST-8)322,327,930
 3201 IF(M-IST-7)322,320,321
  321 IF(M-IST-12)322,327,930
  322 IF(IFF)930,323,319
  323 XNX=XNX*10000.
      NM=2
      IG=1000
      GO TO 318
  324 N=IST+11
      DO 325 M=IST,N
  325 IBF(M)=23616
  327 CONTINUE
      N=IST+11
      DO 3270 M=1,N
      IF((IBF(M)+10)/10-1)3270,3271,3270
 3271 NN=IBF(M)+1
      IBF(M)=ICR(NN)
 3270 CONTINUE
      WRITE(J,329)(IBF(M),M=1,N)
  329 FORMAT(1H ,105A1)
      CALL DATSW(13,ISW)
      GO TO(330,328),ISW
  328 CONTINUE
  330 LM=0
      GO TO 121
  938 IE=8
  930 IE=IE+8
  922 IE=IE+22
  900 RETURN
      END
```

PROGRAM LST

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE LST
      DIMENSION DATA(2)
      DIMENSION IBUF(90),ICR(50),ICPT(95,6),LOOK(24)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (K2,KODE(2)),(K1,KODE(1)),(K3,KODE(3))
      EQUIVALENCE(IBL,ICR(11)),(IQ,IT(10)),(IP,IT(9))
      EQUIVALENCE (ICPT(1,1),KODE(1))
      EQUIVALENCE (KODE(2),DATA(1))
      EQUIVALENCE (NP,KODE(4))
      GO TO(3,3,1),IEX
    1 WRITE(1,2)
    2 FORMAT(41H  ** LIST CANNOT BE USED WHILE EDITING **)
      GO TO 501
    3 CALL DATSW(13,ISW)
      GO TO(501,31),ISW
   31 IISV=II
      IM=II-IP
      II=4591
      WRITE(1'II)(KODE(M),M=1,1140),(IT(N),N=1,26)
      CALL DATSW(12,KKQ)
      CALL DATSW(0,K)
      KQ=5-2*K
      IF(IQ)300,400,69
   69 II=IISV
      READ(3'II)(KODE(K),K=1,IQ)
      I=2*KODE(1)+1
      WRITE(KQ,82)
   82 FORMAT(1H )
      IX=1
      II=501
      READ(5'II)(ICR(K),K=1,50)
   83 J=0
      GO TO(86,101),IX
   86 I=I+1
      K1=KODE(I)/256
      K2=KODE(I)-256*K1
      IF(K1-50)100,105,106
  100 J=J+1
      IBUF(J)=ICR(K1+1)
      IF(K2-50)101,105,105
  101 J=J+1
      IBUF(J)=ICR(K2+1)
      IF(I-IQ)86,110,110
  105 IX=1
      GO TO 110
  106 IX=2
  110 WRITE(KQ,111)(IBUF(K),K=1,J)
  111 FORMAT(8X,80A1)
      GO TO (800,117),KKQ
  800 WRITE(2,801)(IBUF(K),K=1,J)
  801 FORMAT(80A1)
  117 CALL DATSW(13,ISW)
      GO TO(502,1170),ISW
 1170 IF(I-IQ)83,200,200
  200 WRITE(KQ,82)
      CALL DATSW(2,K)
      GO TO (711,712),K
```

```
711 II=IP
    READ(3'II)(KODE(K),K=1,IM)
    DO 701 K=1,7
    WRITE(3,700)KODE(K)
700 FORMAT(6X,I6)
701 CONTINUE
    CALL DATSW(13,ISW)
    GO TO(500,702),ISW
702 DO 703 K=8,K1
    KOP=KODE(K)/512
    KOA=KODE(K)-KOP*512
    WRITE(3,704)KOP,KOA
704 FORMAT(2I6)
703 CONTINUE
    M=K1+1
    CALL DATSW(13,ISW)
    GO TO(500,705),ISW
705 DO 708 K=M,K2
708 WRITE(3,700)KODE(K)
    IF(K3-K2)715,715,714
714 N=(K2+2)/2
    KQ=K3/2
    CALL DATSW(13,ISW)
    GO TO(500,707),ISW
707 DO 710 M=N,KQ
710 WRITE(3,709)DATA(M)
709 FORMAT(F12.4)
715 NN=(IM-K3)/4
    CALL DATSW(13,ISW)
    GO TO(500,713),ISW
713 IF(NN)712,712,716
716 K=K3
    DO 718 N=1,NN
    L1=1
    DO 717 L=1,3
    KQ=K+L
    M=KODE(KQ)/256
    J=KODE(KQ)-256*M
    LOOK(L1)=ICR(M+1)
    LOOK(L1+1)=ICR(J+1)
717 L1=L1+2
    WRITE(3,901)(LOOK(KQ),KQ=1,6)
901 FORMAT(6X,6A1)
718 K=K+4
712 GO TO 500
300 CALL DATSW(2,KX)
    IF (NCP) 301,301,303
301 WRITE(KQ,302)
302 FORMAT(30H THERE ARE NO PROGRAMS DEFINED)
    GO TO 500
303 II=4
    READ(3'II)((ICPT(J,K),K=1,6),J=1,NCP)
    II=501
    READ(5'II)(ICR(K),K=1,50)
    WRITE(KQ,82)
    GO TO(305,304),KX
304 M=(NCP+3)/4
    IP=4
    GO TO 306
305 M=NCP
    IP=1
306 DO 316 N=1,M
    L1=0
    DO 309 NN=1,IP
    MM=N+(NN-1)*M
    IF(MM-NCP)307,307,310
307 DO 308 L=1,3
    L1=L1+2
    J=ICPT(MM,L)/256
    K=ICPT(MM,L)-256*J
    LOOK(L1-1)=ICR(J+1)
308 LOOK(L1)=ICR(K+1)
309 CONTINUE
310 GO TO(311,313),KX
311 WRITE(KQ,312)(LOOK(K),K=1,6),(ICPT(MM,J),J=4,6)
312 FORMAT(9X,6A1,3I8)
    GO TO 315
313 WRITE(KQ,314)(LOOK(K),K=1,L1)
314 FORMAT(9X,6A1,10X,6A1,10X,6A1,10X,6A1)
315 CALL DATSW(13,ISW)
    GO TO(502,316),ISW
316 CONTINUE
502 WRITE(KQ,82)
500 II=4591
    READ(1'II)(KODE(M),M=1,1140),(IT(N),N=1,26)
501 IE=4
    RETURN
400 II=(IP-1)*2+2100
    READ(5'II)ISTT,NCH
    IF(NCH)500,500,401
401 II=ISTT
    READ(5'II)(KODE(K),K=1,NCH)
    M=1
    K=1
    WRITE(KQ,82)
    DO 403 N=1,NCH
    IF(KODE(N)-23387)403,402,403
402 K=N-1
    WRITE(KQ,404)(KODE(J),J=M,K)
    M=N+1
    CALL DATSW(13,ISW)
    GO TO(502,403),ISW
403 CONTINUE
    WRITE(KQ,404)(KODE(J),J=M,NCH)
404 FORMAT(8X,40A2)
    GO TO 502
    END
```

SUBPROGRAM KYBRD

```
      ENT        KYBRD
KYBRD BSS        1
      LDX    I2  KYBRD
      STX     2  ARG+1
      MDX     2  1
      STX     2  RTRN+1
ARG   LD     L   0
      LIBF       TYPAM
RTRN  BSC    L   0
      END
```

SUBPROGRAM TYPAM

```
       LIBR
       ENT         TYPAM
SAV12  BSS         1
SIX    DC          6
COUNT  BSS         1
C10    DC          10
C51    DC          51
TYBLK  DC          /2100
NEWL2  LD          C51
       STO       1 0
       BSI         CKBF
       MDX       1 -1
       STX       1 LBFAD
       LDX       2 6
LOOP1  LD          C10
       STO       1 0
       MDX       1 -1
       BSI         CKBF
       MDX       2 -1
       MDX         LOOP1
       MDX         INTL2
CKDEL  LD          BFCHC
       S           LCHCT
       STO         BFCHC
       MDX  L      DELIN,-1
       MDX         INTL1
       MDX         INTL2
TYPAM  MDX         *+2
       BOSC I      0            RETURN
       STO         BFADD        BUFFER ADDRESS
       LD   L      /000C
       STO         SAV12
       LD          IOCC2
       STO  L      /000C
INTL1  LD          BFADD
       STO         LBFAD
       LD          SIX
       STO         BFCHC        BUF. CHAR. COUNT
INTL2  LD          ZERO
       STO         LCHCT        LINE CHAR. COUNT
REWT   LD          ZERO
       STO         DELIN        $/$$ INDICATOR
       LD          LCHCT        LINE. CHAR. COUNT
       A           SIX
       STO         COUNT
       LD          LBFAD
       STO  L      1
       LDX       2 2
       LD          TWO
       S    L      /75E9
       BSC         +
       LDX       2 1
       LD          SELCT
       BSI         STO
       MDX       2 -1
       MDX         LOOP
       LDX       2 8
EDIT   LD          TYBLK
       BSI         STO
```

```
      MDX    2 -1
      MDX      EDIT
LOOP  LD     1 0
      STO   L  2
      BSI      GETTY
      MDX   L  COUNT,-1
      MDX      LOOP
      MDX      READ
CKBF  DC       0
      MDX   L  BFCHC,1
      LD       BFCHC
      S        C297
      BSC   I  CKBF,+
      MDX      RESET
LBFAD DC       0              LINE BUFFER ADDRESS
BFCHC DC       0              BUFFER CHAR. COUNT
C50   DC       50
EOLS  DC       0              EOL/EOS
C297  DC       297
BFADD DC       0              BUFFER ADDRESS
TWO   DC       2
INTRP DC       0              INTERRUPT ROUTINE
      XIO      SENSE
      BOSC I   STO,+Z
      XIO      IOCC           READ CHARACTER
      LD     1 0
      SLA      12
      BOSC L   CHAR,-+        DATA CHARACTER
      SLA      1              CHECK FOR CARR. RETURN
      BOSC L   CHAR2,-Z
RESET STO      EOLS
CKLIN LD       DELIN
      BOSC L   CKDEL,Z-
      BOSC L   REWT,Z
      LD       EOLS
      BOSC L   NEWL2,Z+       BRANCH IF CR.
      BOSC L   EOS,+
EOF   LD       C50
      STO    1 0
      LD       BFCHC
      A        ONE
SAVE  STO   L  /75F4
      LD       SAV12
      STO   L  /000C
      MDX      TYPAM+1
EOS   MDX   L  BFCHC,1
      LDX    2 45
      STX   I2 1
      BSI      GETTY
      MDX      EOF
      BSS   E  0
IOCC  DC       0
      DC       /0A00
SELCT DC       /8100
      DC       /0C00
IOCC2 DC       INTRP
      DC       /0900
SENSE DC       0
      DC       /0F01
```

```
LCHCT DC        0              LINE CHAR. COUNT
ZERO  DC        0
ONE   DC        1
MONE  DC        -1
DELIN DC        0              $/$$
STO   BSS       1
      STO       INTRP
      BSI       TNRDY
      XIO       IOCC2
RDWRT WAIT
      MDX       *-2
CHAR  LDX    2  51
CHAR1 LD    L2  HOLTB
      EOR    1  0
      BSC   L   MATCH,+-
      MDX    2  -1
      MDX       CHAR1
CHAR2 LDX    2  11
MATCH MDX    2  -1
      MDX       *
      STX   I2  1
      BSI       GETTY
TYPE1 LD     1  1
      S         C48
      BSC   L   DEL,+-
      S         TWO
      BSC   L   BKSP,+-
UPDT  MDX   L   LCHCT,1
      BSI       CKBF
      LD        LCHCT
      S         C74
      BSC   L   NEWL2,-
READ  STX    1  IOCC
      BSI       TNRDY
      XIO       SELCT
      MDX       RDWRT
GETTY BSS       1
      MDX    1  -1
      LD    L   2
      SRT       1
      STO   L   2
      LD        ZERO
      SLT       1
      BSC   L   SEC,Z
      LD    L2  TYPTB
      SRA       8
      MDX       SAME
SEC   LD    L2  TYPTB
      SRT       8
      LD        ZERO
      SLT       8
SAME  SLA       8
      BSI       STO
      BSC   I   GETTY
DEL   LD     1  2
      EOR       C48
      BSC       Z
      LD        MONE
      A         TWO
      STO       DELIN
      MDX       UPDT
BKSP  LD        DELIN
      BSC       +-
      S         ONE
      STO       DELIN
      MDX     1 1
      LD        LCHCT
      S         ONE
      BSC    L  READ,+Z
      STO       LCHCT
      MDX    L  BFCHC,-1
      MDX     1 1
      LD      1 0
      EOR       C48
      BSC    L  READ,Z
      LD        DELIN
      S         ONE
      BSC       +-
      S         ONE
      STO       DELIN
      MDX       READ
C48   DC        48
C74   DC        74
TNRDY DC        *-*
      XIO       SENSE
      SLA       5
      BSC    I  TNRDY,-
      WAIT
      MDX       TNRDY+1
TYPTB DC        /C4FC       0,1
      DC        /D8DC       2,3
      DC        /F0F4       4,5
      DC        /D0D4       6,7
      DC        /E4E0       8,9
      DC        /213C       BLANK,A
      DC        /181C       B,C
      DC        /3034       D,E
      DC        /1014       F,G
      DC        /2420       H,I
      DC        /7C58       J,K
      DC        /5C70       L,M
      DC        /7450       N,O
      DC        /5464       P,Q
      DC        /6098       R,S
      DC        /9CB0       T,U
      DC        /B490       V,W
      DC        /94A4       X,Y
      DC        /A0D6       Z,*
      DC        /BCDA       /,+
      DC        /8444       -,&
      DC        /C2FE       =,(
      DC        /F600       ),.
      DC        /80D2       ,,SEMICOLON
      DC        /40E6       $,'
      DC        /C000
HOLTB EQU       *-1
      DC        /2000       0
      DC        /1000       1
DC        /0800        2
DC        /0400        3
DC        /0200        4
DC        /0100        5
DC        /0080        6
DC        /0040        7
DC        /0020        8
DC        /0010        9
DC        0
DC        /9000        A
DC        /8800        B
DC        /8400        C
DC        /8200        D
DC        /8100        E
DC        /8080        F
DC        /8040        G
DC        /8020        H
DC        /8010        I
DC        /5000        J
DC        /4800        K
DC        /4400        L
DC        /4200        M
DC        /4100        N
DC        /4080        O
DC        /4040        P
DC        /4020        Q
DC        /4010        R
DC        /2800        S
DC        /2400        T
DC        /2200        U
DC        /2100        V
DC        /2080        W
DC        /2040        X
DC        /2020        Y
DC        /2010        Z
DC        /4220        *
DC        /3000        /
DC        /80A0        +
DC        /4000        -
DC        /8000        &
DC        /00A0        =
DC        /8120        (
DC        /4120        )
DC        /8420        .
DC        /2420        ,
DC        /40A0        SEMICOLON
DC        /4420        $
DC        /0120        '
DC        /0420
END
```

SUBPROGRAM SERCH

```
       ENT        SERCH
SERCH  DC         0
       LDX   I2   SERCH
       LD     2   0
       STO        TAB+1
       LD     2   1
       STO        COL+1
       LD     2   2
       STO        LIM+1
       MDX    2   3
       STX    2   RTN+1
       LD         ZRO
       STO   L    L
NOM    LD         ZRO
       S     L    L
       STO   L    2
       LDX    1   3
LPO1   MDX   L    L,1
LIM    LD    L    0
       A          ONE
       S     L    L
       BSC   L    LPO2,Z
       STO   L    L
       MDX        RTN
LPO2   LD    L1   LOOK
TAB    EOR   L2   0
       BSC   L    NOM,Z
       LD    L    2
COL    S     L    0
       STO   L    2
       MDX    1   -1
       MDX        LPO2
RTN    BSC   L    0
L      EQU        /75F3
LOOK   EQU        /7D1C
ZRO    DC         0
ONE    DC         1
       END
```

SUBPROGRAM AJS

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE AJS  (LA,LTO,LTN)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE(JS,IT(9)),(JN,IT(10))
      IF(LA)930,3,1
    1 IF (LTO-LTN) 2,900,4
    2 JS=LA
      JN=LTO
      CALL FRE
      IF(IE)900,3,900
    3 LA=0
      LTO=0
      IF (LTN)930,900,30
   30  JN=LTN
      CALL GET
      IF(JS)900,900,5
    4 JS=LA+LTN
      JN=LTO-LTN
      CALL FRE
      IF(IE)900,40,900
   40 IF(LTN)930,41,6
   41 LA=0
      LTO=0
      RETURN
    5 LA=(JS+1)/2
    6 LTO=LTN
  900 RETURN
  930 IE=30
      RETURN
      END
```

SUBPROGRAM GET

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE GET
      DIMENSION ISP(2)
      DIMENSION IPP(2)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      EQUIVALENCE (JS,IT(9)),(JN,IT(10))
      EQUIVALENCE (ISP(1),KODE(2)),(IPP(1),KODE(1))
      EQUIVALENCE(IPPX,IPP(1141))
      IF (JN) 926,926,1
    1 IF(LNT-JN)929,102,102
  102 K=1
      IF (IPPX) 935,929,2
    2 JS=1141
    3 M=JS
      JS=IPP(M)
      IF (JS) 935,8,5
    5 IF(ISP(M)-JN)3,7,6
    6 ISP(M)=ISP(M)-JN
      IPP(M)=2*JN+JS
      M=IPP(M)
    7 IPP(M)=IPP(JS)
      ISP(M)=ISP(JS)
      LNT=LNT-JN
      RETURN
    8 GO TO (9,15,929),K
    9 JS=1141
   10 M=JS
      JS=IPP(JS)
   11 N=IPP(JS)
      IF (N) 935,14,12
   12 IF (N-JS-2*ISP(M)) 935,13,10
   13 ISP(M)=ISP(M)+ISP(JS)
      IPP(JS)=IPP(N)
      ISP(JS)=ISP(N)
      K=2
      GO TO 11
   14 GO TO (15,2),K
   15 CALL GARB
      K=3
      IF(JS)900,900,2
  935 IE=6
  929 IE=IE+3
  926 IE=IE+26
      JS=-1
  900 RETURN
      END
```

SUBPROGRAM GARB

```
        ENT        GARB
IDS     EQU        /76B4
LNT     EQU        /75ED
IDL     EQU        /765A
ISPX    EQU        /7B8A
IPPX    EQU        /7B8B
IT9     EQU        /75F7
IE      EQU        /75F0
IFT     EQU        /75EF
ISP     EQU        /7FFE
IPP     EQU        /7FFF
IDSP    DC         IDS
IDLP    DC         IDL
TEMP    BSS        1
C573    DC         573
N       BSS        1
M       BSS        1
ONE     DC         1
ZRO     DC         0
KK      BSS        1
GARB    BSS        1
        LD     L   IFT
        A          ONE
        STO        KK
        CALL       SHF
        DC         KK
        DC         C90
        LD         C573
        STO        M
        LD         ZRO
        STO        N
        S          KK
        STO    L   1
L150    LD     L1  IDS
        BSC    L   L930,+Z
        BSC    L   L151,Z
        MDX    L   N,1
L151    MDX     1  1
        MDX        L150
L016    LD         ZRO
        STO        TEMP
        S          KK
        STO    L   1
L017    LD     L1  IDS
        BSC    L   L021,+-
        BSC    L   L930,+
        S          M
        BSC    L   L021,+Z
        BSC    L   L023,+
        LD         TEMP
        BSC    L   L020,-
        STO    L   2
        LD     L2  IDS
        S      L1  IDS
        BSC    L   L021,Z+
        BSC    L   L930,+
L020    STX     1  TEMP
L021    MDX     1  1
        MDX        L017
        LD         TEMP
```

```
      BSC  L  L930,-
      A       IDSP
      STO     ARG1
      LD      IDLP
      A       TEMP
      STO     ARG2
      CALL    MOV
      DC      M
ARG1  DC      0
ARG2  DC      0
      LD      TEMP
      STO  L  1
      LD      M
      STO  L1 IDS
L023  LD      M
      A    L1 IDL
      STO     M
      LD      N
      A       ONE
      STO     N
      S       KK
      BSC  L  L016,+Z
      BSC  L  L930,Z
      LD      M
      SLA     1
      S       ONE
      STO  L  IT9
      STO  L  IPPX
      LD      C1177
      S       M
      STO  L  ISPX
      S    L  LNT
      BSC  L  L949,Z
      LD      ONE
      S    L  IT9
      STO  L  1
      LD      ZRO
      STO  L1 IPP
      STO  L1 ISP
      CALL    SHF
      DC      C90
      DC      KK
RTN   BSC  I  GARB
L949  MDX  L  IE,19
L930  MDX  L  IE,30
      LD      MONE
      STO  L  IT9
      MDX     RTN
MONE  DC      -1
C90   DC      90
C1177 DC      1177
      END
```

SUBPROGRAM FRE

```
      ENT       FRE
FRE   BSS       1
      LD    L   N
      S         C572
      BSC   L   L916,+
L001  A     L   K
      S         C605
      BSC   L   L916,-Z
L002  LD    L   N
      SLA       1
      S         ONE
      STO       JA
      LD        ONE
      S         JA
      STO   L   2
      LD        C1141
      STO       KK
L009  LD        ONE
      S         KK
      STO   L   1
      LD    L1  IPP
      STO       KK
      BSC   L   L915,+Z
      BSC   L   L012,+
L011  S         JA
      BSC   L   L009,Z+
      BSC   L   L916,+
L012  LD        KK
      STO   L2  IPP
      LD    L1  ISP
      STO   L2  ISP
      LD        JA
      STO   L1  IPP
      LD    L   K
      STO   L1  ISP
      LD    L   LNT
      A     L   K
      STO   L   LNT
RTN   BSC   I   FRE
L916  MDX   L   IE,1
L915  MDX   L   IE,35
      MDX       RTN
C572  DC        572
C605  DC        605
C1141 DC        1141
ONE   DC        1
KK    DC        0
JA    DC        0
N     EQU       /75F7
K     EQU       /75F6
ISP   EQU       /7FFE
IPP   EQU       /7FFF
IE    EQU       /75F0
LNT   EQU       /75ED
      END
```

SUBPROGRAM MOV

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE MOV (LA,LB,LC)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      IF(LA-LB)2,3,2
    2 LD=2*LA-1
      M=2*LB-1
      M=2*LB-1
      N=2*(LB+LC-1)
      DO 1 K=M,N
      KODE(LD)=KODE(K)
    1 LD=LD+1
    3 RETURN
      END
```

SUBPROGRAM SHF

```
*LIST ALL
*ONE WORD INTEGERS
      SUBROUTINE SHF  (LA,LB)
      COMMON KODE(2380),IDAT(90,2),IT(10),
     1I,J,L,II,ID,IE,IFT,KON,LNT,LPV,NCP,NV,IEX,IGT,NMB,NAP
      DO 1 K=1,2
      M=IDAT(LA,K)
      IDAT(LA,K)=IDAT(LB,K)
    1 IDAT(LB,K)=M
      RETURN
      END
```

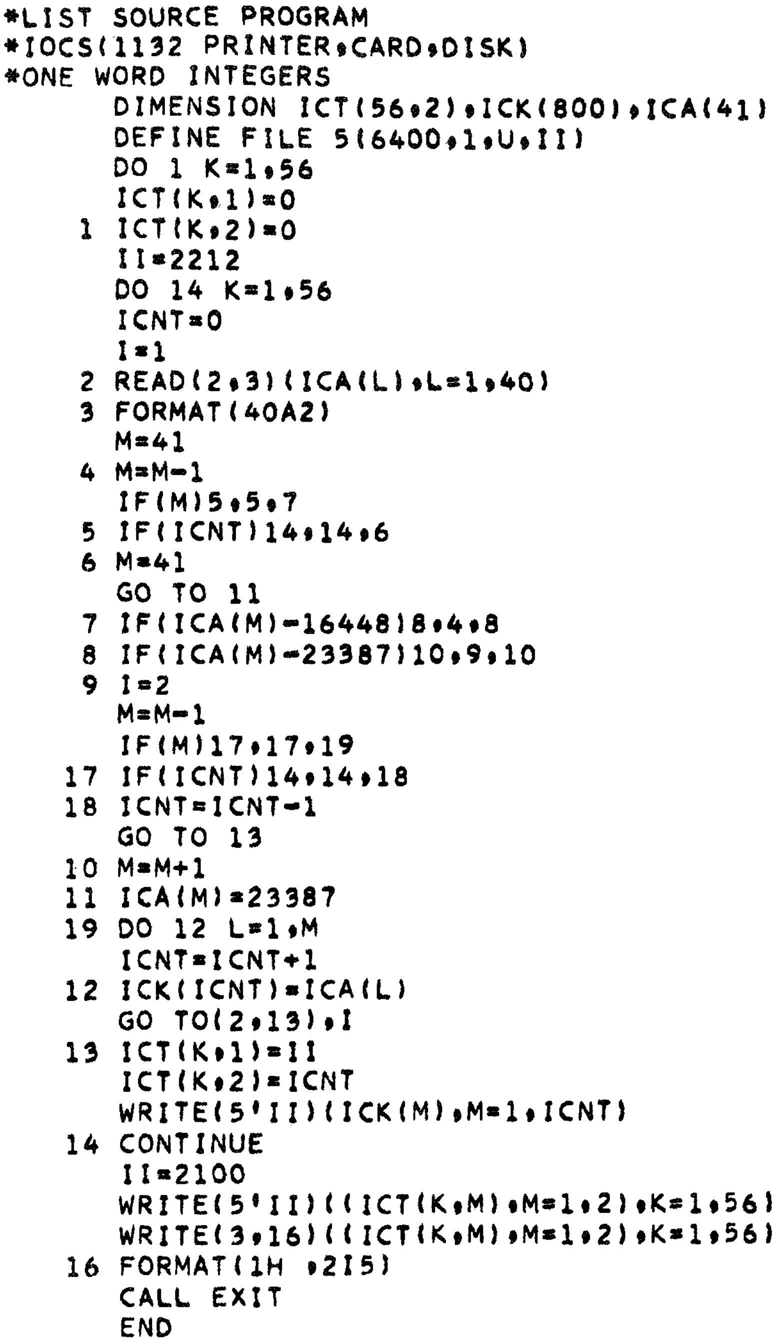
PROGRAM INTL3

```
*LIST SOURCE PROGRAM
*IOCS(1132 PRINTER,DISK)
*ONE WORD INTEGERS
      DIMENSION INT(3)
      DEFINE FILE 3(30720,1,U,II)
      INT(1)=0
      INT(2)=580
      INT(3)=0
      II=1
      WRITE(3'II)INT
      II=1
      READ(3'II)INT
      WRITE(3,1)(INT(K),K=1,3)
    1 FORMAT(1H ,3I5)
      CALL EXIT
      END
```

PROGRAM ALT5

```
*LIST ALL
*ONE WORD INTEGERS
*IOCS(DISK,1132 PRINTER,CARD)
      DEFINE FILE 5(6400,1,U,II)
      DIMENSION IOP(100,4),KEEP(6),ICR(50),ITB(100),KCH(40)
      DIMENSION LRPT(15)
      READ(2,30)(ICR(I),I=1,50)
   30 FORMAT(50A1)
      CALL DATSW (10,MZZ)
      GO TO (800,500),MZZ
  800 II=1
      READ(5'II)((IOP(J,K),K=1,4),J=1,100)
      GO TO 99
  500 CONTINUE
      DO 29 M=1,100
      DO 29 N=1,3
   29 IOP(M,N)=2570
   99 READ(2,100)IPN,(KEEP(K),K=1,6),IP4
  100 FORMAT(I3,2X,6A1,8X,I4)
      WRITE(3,100)IPN,(KEEP(K),K=1,6),IP4
      IF (IPN) 120,120,101
  101 DO 104 K=1,6
      LL=KEEP(K)
      DO 103 N=1,37
      IF (LL-ICR(N)) 103,102,103
  102 KEEP(K)=N-1
      GO TO 104
  103 CONTINUE
      WRITE(3,1035)
 1035 FORMAT(38H ******THE ABOVE NAME IS INVALID******)
      GO TO 99
  104 CONTINUE
      DO 105 M=1,3
      MK=M+M
  105 IOP(IPN,M)=256*KEEP(MK-1)+KEEP(MK)
      IOP(IPN,4)=IP4
      GO TO 99
  120 II=1
      WRITE(5'II)((IOP(J,K),K=1,4),J=1,100)
      GO TO (11,550),MZZ
  550 CONTINUE
      CALL DATSW(0,K)
      GO TO(31,32),K
   31 II=1
       READ(5'II)((IOP(J,K),K=1,4),J=1,100)
       WRITE(3,1)((IOP(J,K),K=1,4),J=1,60)
    1 FORMAT(4I6)
   32 LRPT(1)=430
      LRPT(2)=264
      LRPT(3)=386
      LRPT(4)=268
      LRPT(5)=430
      LRPT(6)=264
      LRPT(7)=430
      LRPT(8)=261
      LRPT(9)=387
      LRPT(10)=268
      LRPT(11)=218
      LRPT(12)=265
      LRPT(13)=430
      LRPT(14)=262
      LRPT(15)=265
      READ(2,30)(KCH(K),K=18,34),(KCH(K),K=5,17)
      KCH(1)=1
      KCH(2)=10
      KCH(3)=100
      KCH(4)=1000
      II=401
      WRITE(5'II)(LRPT(I),I=1,15)
      II=467
      WRITE(5'II)(KCH(K),K=1,34),(ICR(K),K=1,50)
      II=701
      DO 2 K=1,100
    2 ITB(K)=0
      CALL DATSW(1,IDS)
      DO 9 J=1,70
      READ(2,3)(KCH(K),K=1,40)
    3 FORMAT(40A2)
      DO 4 K=1,40
      IF(KCH(K)-16448)4,5,4
    4 CONTINUE
    5 K=K-1
      ITB(J)=II
      IF(K)8,8,6
    6 WRITE(5'II)(KCH(M),M=1,K)
      GO TO(60,9),IDS
   60 WRITE(3,7)J,(KCH(M),M=1,40),ITB(J)
    7 FORMAT(1H ,I2,2H  ,40A2,2H  ,I5)
      GO TO 9
    8 GO TO(80,9),IDS
   80 WRITE(3,7)J
    9 CONTINUE
      ITB(J)=II
      II=601
      WRITE(5'II)(ITB(K),K=1,100)
   11 CALL EXIT
      END
```

PROGRAM ALT5A



```
*LIST SOURCE PROGRAM
*IOCS(1132 PRINTER,CARD,DISK)
*ONE WORD INTEGERS
      DIMENSION ICT(56,2),ICK(800),ICA(41)
      DEFINE FILE 5(6400,1,U,II)
      DO 1 K=1,56
      ICT(K,1)=0
    1 ICT(K,2)=0
      II=2212
      DO 14 K=1,56
      ICNT=0
      I=1
    2 READ(2,3)(ICA(L),L=1,40)
    3 FORMAT(40A2)
      M=41
    4 M=M-1
      IF(M)5,5,7
    5 IF(ICNT)14,14,6
    6 M=41
      GO TO 11
    7 IF(ICA(M)-16448)8,4,8
    8 IF(ICA(M)-23387)10,9,10
    9 I=2
      M=M-1
      IF(M)17,17,19
   17 IF(ICNT)14,14,18
   18 ICNT=ICNT-1
      GO TO 13
   10 M=M+1
   11 ICA(M)=23387
   19 DO 12 L=1,M
      ICNT=ICNT+1
   12 ICK(ICNT)=ICA(L)
      GO TO(2,13),I
   13 ICT(K,1)=II
      ICT(K,2)=ICNT
      WRITE(5'II)(ICK(M),M=1,ICNT)
   14 CONTINUE
      II=2100
      WRITE(5'II)((ICT(K,M),M=1,2),K=1,56)
      WRITE(3,16)((ICT(K,M),M=1,2),K=1,56)
   16 FORMAT(1H ,2I5)
      CALL EXIT
      END
```

PROGRAM HOLTB

```
      LIBR
      ENT    HOLTB
      DC     0       BLANK
      DC     /2000   0
      DC     /1000   1
      DC     /0800   2
      DC     /0400   3
      DC     /0200   4
      DC     /0100   5
      DC     /0080   6
      DC     /0040   7
      DC     /0020   8
      DC     /0010   9
      DC     /4000   -
      DC     /8420   .
      DC     /8100   E
      DC     /80A0   +
      DC     /8000   AMPERSAND
      DC     /00A0   =
      DC     /4120   )
      DC     /8120   (
      DC     /2420   ,
      DC     /0120   APOSTROPHE
      DC     /3000   /
      DC     /4220   *
      DC     /4420   $
      DC     /8220   LESS THAN
      DC     /40A0   SEMI COLON
      DC     /0420   POUND SIGN
      DC     /0220   AT SIGN
      DC     /9000   A
      DC     /8800   B
      DC     /8400   C
      DC     /8200   D
      DC     /8080   F
      DC     /8040   G
      DC     /8020   H
      DC     /8010   I
      DC     /5000   J
      DC     /4800   K
      DC     /4400   L
      DC     /4200   M
      DC     /4100   N
      DC     /4080   O
      DC     /4040   P
      DC     /4020   Q
      DC     /4010   R
      DC     /2800   S
      DC     /2400   T
      DC     /2200   U
      DC     /2100   V
      DC     /2080   W
      DC     /2040   X
      DC     /2020   Y
      DC     /2010   Z
X     DC     0
HOLTB EQU    X-2
      END
```

PROGRAM EBCTB

```
LIBR
ENT     EBCTB
DC      /1611    BACKSPACE
DC      /1581    CARRIER RETURN
DC      /2503    LINE FEED
DC      /1405    SHIFT TO PRINT BLACK
DC      /3509    SHIFT TO PRINT RED
DC      /0541    TABULATE
DC      /4021    BLANK
DC      /F0C4    0
DC      /F1FC    1
DC      /F2D8    2
DC      /F3DC    3
DC      /F4F0    4
DC      /F5F4    5
DC      /F6D0    6
DC      /F7D4    7
DC      /F8E4    8
DC      /F9E0    9
DC      /6084    -
DC      /4B00    .
DC      /C534    E
DC      /4EDA    +
DC      /5044    AMPERSAND
DC      /7EC2    =
DC      /5DF6    )
DC      /4DFE    (
DC      /6B80    ,
DC      /7DE6    APOSTROPHE
DC      /61BC    /
DC      /5CD6    *
DC      /5B40    $
DC      /4CDE    LESS THAN
DC      /5ED2    SEMICOLON
DC      /7BC0    POUND SIGN
DC      /7C04    AT SIGN
DC      /C13C    A
DC      /C218    B
DC      /C31C    C
DC      /C430    D
DC      /C610    F
DC      /C714    G
DC      /C824    H
DC      /C920    I
DC      /D17C    J
DC      /D258    K
DC      /D35C    L
DC      /D470    M
DC      /D574    N
DC      /D650    O
DC      /D754    P
DC      /D864    Q
DC      /D960    R
DC      /E298    S
DC      /E39C    T
DC      /E4B0    U
DC      /E5B4    V
DC      /E690    W
DC      /E794    X
```

```
      DC      /E8A4     Y
      DC      /E9A0     Z
X     DC      0
EBCTB EQU     X-2
      END
```